UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

APLICACIÓN PARA LA GESTIÓN Y ADMINISTRACIÓN DE LAS PEQUEÑAS Y MEDIANAS EMPRESAS (PYMES)

Jaime Constanzo Cedillo

Tutor: Roberto Latorre

Ponente: Ana Mª González

Julio de 2014

RESUMEN

En este proyecto se pretende desarrollar una herramienta que permita facilitar y automatizar los procesos que una PYME (Pequeña y Mediana Empresa) debe realizar para garantizar una gestión y administración de la empresa correcta, rápida y fácil. En particular, dado que resulta muy complejo desarrollar una herramienta sin conocer procesos y datos reales, el proyecto se ha desarrollado atendiendo a las necesidades y características de un cliente en concreto. La empresa analizada se caracteriza por ser una empresa de construcción cuyo negocio reside en la compra de productos al por mayor, los cuales, son envasados, vendidos y distribuidos al por menor.

Durante la realización del proyecto se han realizado continuas reuniones con el cliente para, en un primer momento, capturar sus necesidades, y posteriormente, poder realizar un análisis, diseño e implementación de la aplicación atendiendo, en la medida de lo posible, a sus gustos y opiniones. El cliente ha jugado un papel muy importante durante el desarrollo del proyecto.

La herramienta desarrollada pretende ser una BPMS (Business Process Management Suite) con características propias de un ERP (Enterprise Resource Planning), un CRM (Customer Relationship Management) y un TPV (Terminal Punto de Venta), aunando en un mismo sistema las funcionalidades que ofrecen estas herramientas, como la gestión de la producción, distribución, inventario, contabilidad, facturas, ventas, cartera de clientes y productos, etc. Entre las posibles mejoras de la herramienta desarrollada reside la inclusión de funcionalidad de cálculo de rutas (empleando Google Maps) o el cálculo de los estados financieros de la empresa.

PALABRAS CLAVE

Software para empresas, PYME, automatización de procesos, gestión, administración, BPMS, TPV, ERP, CRM, Java, modelo de desarrollo incremental, MVC, MyBatis, DAO.

ABSTRACT

The aim of this project is to develop a tool to facilitate and automate the tasks that must be carried out by a SME (Small and Medium Business) in order to ensure a successful, fast and easy management and administration of the company. Due to it is difficult to develop a tool without knowing real processes and real data, the project has been developed according to the requirements and characteristics of a specific customer.

During this project there have been regular meetings with the customer to capture his needs and educe the application requirements and, then, to analyze, design and implement the application considering, as far as possible, his opinions. The customer has played an important role during the project development.

The software tool implemented has been developed with the idea of being a BPMS (Business Process Management Suite) with characteristics of an ERP (Enterprise Resource Planning), CRM (Customer Relationship Management) and POS (Point of Sale), combining in a single system the features that these tools offer such as production management, distribution, inventory, accounting, billing, sales, client portfolio, products catalog, etc. The tool may be improved in the future by including route calculation functionality (using Google Maps) or giving the financial statements of the company.

KEY WORDS

Industrial software, SMEs, process automation, management, administration, BPMS, POS, ERP, CRM, Java, incremental development model, MVC, MyBatis, DAO.

GLOSARIO

API *(**A**pplication **P**rogramming **I**nterface)*
Conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

ATDD *(**A**cceptance **T**est-**D**riven **D**evelopment)*
Metodología de desarrollo basada en la comunicación entre cliente, desarrollador y tester.

BEAN Componente software que tiene la particularidad de ser reutilizable y así evitar la tediosa tarea de programar los distintos componentes uno a uno.

BPMS *(**B**usiness **P**rocess **M**anagement **S**uite)*
Conjunto de servicios y herramientas que facilitan la administración de procesos de negocio.

CRM *(**C**ustomer **R**elationship **M**anagement)*
Sistemas informáticos de apoyo a la gestión de las relaciones con los clientes, a la venta y al marketing.

ERM *(**E**nterprise **R**isk **M**anagement)*
Herramienta que incluye los métodos y procesos utilizados por las organizaciones para gestionar los riesgos y aprovechar las oportunidades relacionadas con el logro de sus objetivos.

ERP *(**E**nterprise **R**esource **P**lanning)*
Sistemas de información gerenciales que integran y manejan muchos de los negocios asociados con las operaciones de producción y de los aspectos de distribución de una compañía en la producción de bienes o servicios.

DAO *(**D**ata **A**ccess **O**bject)*
Componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo.

JAR *(**J**ava **AR**chive)* – Siglas escogidas para coincidir con "jar" (Tarro)
Tipo de archivo que permite ejecutar aplicaciones escritas en el lenguaje Java.

JAVA Lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible.

JDK *(**J**ava **D**evelopment **K**it)*
Software que provee herramientas de desarrollo para la creación de programas en Java.

MVC *(**M**odelo **V**ista **C**ontrolador)*
Patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.

MyBATIS Herramienta de persistencia Java que se encarga de mapear sentencias SQL y procedimientos almacenados con objetos a partir de ficheros XML o anotaciones.

MySQL Sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones.

PYME *(**P**equeña **Y** **M**ediana **E**mpresa)*
Empresa mercantil, industrial, etc., compuesta por un número reducido de trabajadores, y con un moderado volumen de facturación.

SQL *(**S**tructured **Q**uery **L**anguage)*
Lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas.

TPV *(**T**erminal **P**unto de **V**enta)*
Dispositivo y tecnologías que ayudan en la tarea de gestión de un establecimiento comercial de venta al público que puede contar con sistemas informáticos especializados mediante una interfaz accesible para los vendedores.

XML *(e**X**tensible **M**arkup **L**anguage)*
Lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. MOTIVACIÓN

La motivación de este TFG (Trabajo de Fin de Grado) surge de la necesidad de un cliente para automatizar los procesos administrativos y de gestión de su empresa. Esta automatización, en el ámbito profesional y/o empresarial, se traduce en una mayor calidad de los procesos, así como un ahorro en tiempo, y por lo tanto, en dinero.

La gestión y administración de una empresa es una tarea ardua y difícil, ya que son muchos los procesos que se deben llevar a cabo y muy grande la cantidad de tiempo que se debe destinar a ello si se desea realizar de una manera correcta.

A día de hoy, existen multitud de herramientas, así como empresas que las desarrollan, que facilitan la realización de estas tareas. Típicamente, este tipo de herramientas pueden ser de tres tipos:

- **A medida:** software específico que se adapta a los procesos y necesidades de una única empresa. Estas herramientas son desarrolladas para un cliente concreto y por lo tanto, suelen ser mucho más caras. Generalmente van acompañadas de un contrato de mantenimiento con la empresa que lo desarrolla.

- **Genéricas:** son sistemas cerrados que suelen estar predefinidos por sectores de comercio, por ejemplo, TPVs para las tiendas de ropa, hostelería, videoclubes, etc. Son mucho más baratos que los sistemas a medida pero a cambio no son aplicaciones personalizadas y desarrolladas específicamente para cubrir por completo un conjunto concreto de necesidades o requisitos.

- **Genéricas configurables**: son un tercer tipo de herramientas a medio camino entre las herramientas a medida y las herramientas genéricas. Este tercer tipo se caracteriza por ser aplicaciones que cubren necesidades propias de una herramienta a medida con costes más similares a las herramientas genéricas. El problema que tienen reside en que se necesitan conocimientos expertos para configurarlas de manera que cubran las necesidades concretas del cliente. Podemos decir que son sistemas genéricos que pueden requerir un proceso, avanzado y complejo, de configuración a medida.

En el apartado 2.2 se evaluarán algunas de estas herramientas para desarrollar una solución basada en algunas de sus características.

Dado que la mayor parte de las herramientas genéricas existentes no abarcan en una única aplicación toda la variedad funcional que una PYME podría necesitar, surge la motivación de realizar un desarrollo ad-hoc que se ajuste al detalle a las necesidades exactas del cliente. Existen herramientas de gestión que hacen las veces de un software BPMS pero que no cuentan con las características que un TPV ofrece (facilidad de uso y rapidez), o por el contrario, existen herramientas que ofrecen una funcionalidad mucho mayor que las necesidades que el cliente tiene para una característica concreta.

Para situarnos mejor en el contexto del cliente, cabe decir que se trata de una empresa de construcción cuyo principal volumen de negocio se basa en el envasado y la venta de arena y otros materiales de construcción y jardinería. A groso modo, los procesos que se realizan en la empresa día a día son:

- Compra de materiales al por mayor.
- Envasado (Producción).
- Venta al por menor.
- Servicio de distribución.

A estos procesos de negocio se deben añadir todos los procesos que la gestión de una empresa implica, como el pago de las nóminas y la seguridad social, gastos por luz y agua, impuestos, etc., los cuales se registran en la aplicación como gastos fijos o variables, según corresponda.

1.2. OBJETIVOS

Uno de los principales objetivos establecidos para este TFG reside en saber aplicar en el mundo real los conceptos de análisis, diseño y desarrollo de software aprendidos durante el grado, y en especial, en asignaturas como *Estructuras de Datos*, *Sistemas Informáticos*, *Análisis y Desarrollo de Software* (y proyecto) o *Ingeniería del Software* (y proyecto).

Con este trabajo se pretende desarrollar una herramienta que sea capaz de gestionar todos los procesos que se realizan en el día a día de la empresa de manera unificada, rápida y sencilla. Para ello, y como ya se ha mencionado, resulta de vital importancia la estrecha colaboración del cliente para poder desempeñar esta labor de manera correcta.

La principal diferencia con las prácticas realizadas durante el grado reside en saber atender a las necesidades de un cliente (y no acogiéndonos al marco preestablecido en los enunciados de las prácticas), y a partir de una idea, ser capaz de "pulirla" y desarrollarla hasta lograr lo que el cliente quiere.

Los objetivos del proyecto son:

- Conocer en detalle los procesos de negocio de la empresa para poder automatizarlos de una manera eficiente y que las actividades que se realicen con la aplicación resulten familiares e intuitivas para el usuario.

- Recopilar todos los datos que actualmente se utilizan o registran (ya sea a papel o mediante hojas de cálculo sencillas) así como los que se desee incorporar en la nueva plataforma, para que el diseño del modelo de datos sea lo más completo posible. Es muy importante que el modelo de datos sea completo ya que será quien sostenga toda la funcionalidad implementada en la aplicación, y cualquier resquicio en el mismo puede ocasionar una gran pérdida de tiempo cuando se implemente el resto de la herramienta.

- Analizar los requisitos educidos, determinar la viabilidad de los mismos y realizar un diseño y planificación que sea capaz de satisfacerlos a la vez que permita poder ampliar y mantener en un futuro la herramienta de la manera menos costosa posible.

- Desarrollar la aplicación en cuestión a la vez que se ejecuta un plan de pruebas que verifique y valide las diferentes funcionalidades implementadas en la herramienta. En el apartado 2.5 veremos por qué es importante realizar el desarrollo y las pruebas de manera conjunta.

- Plantear futuras mejoras así como un plan de mantenimiento.

Todos estos puntos se desarrollarán más detenidamente en los siguientes apartados del documento.

Por último, se debe indicar que aunque el proyecto sobre el que se desarrolla este TFG abarca la completa implementación e implantación de un sistema software en varios incrementos (en el apartado 2.5 se verá por qué se habla de incrementos), en la presente memoria se recoge sólo una parte del proceso, ya que el tiempo estimado para la totalidad del mismo sobrepasa las 300 horas que precisa un TFG.

1.3. ESTRUCTURA DEL DOCUMENTO

Una vez se decidió desarrollar una herramienta a medida, se definió una metodología de trabajo basada en los ciclos de vida software estudiados en las asignaturas de *Análisis y Desarrollo de Software* e *Ingeniería del Software*.

La memoria del presente TFG se ha estructurado en base a las fases de la metodología empleada:

- **Análisis:** recoge el proceso de toma y análisis de requisitos así como un estudio de las herramientas existentes y una comparativa, atendiendo a los requisitos educidos, entre dichas herramientas y la herramienta desarrollada. Además, se determina el ciclo de vida software para el proyecto desarrollado.

- **Diseño y Codificación:** se detallan las decisiones de diseño tomadas tras la fase de análisis, tales como la arquitectura y tecnologías empleadas, los patrones de diseño utilizados, etc.

- **Pruebas:** contiene toda la información relativa a las diferentes pruebas realizadas sobre la aplicación.

- **Puesta en producción / Instalación:** describe qué pasos se deben realizar para la correcta puesta en producción del producto y el manual de usuario para formarle en el uso de la aplicación.

- **Futuras ampliaciones y mantenimiento:** explica qué funcionalidad se podría incorporar en futuras versiones de la aplicación y cuál es el plan de mantenimiento a realizar sobre el producto desarrollado.

- **Conclusiones:** extraídas tras la realización del proyecto.

2. ANÁLISIS Y ESTADO DEL ARTE

Para poder hacer un mejor estudio y comparativa de las características de las herramientas existentes así como de las aportadas por la herramienta desarrollada, se expondrán primero los requisitos educidos de las necesidades del cliente para poder enfocar dichas características como ventajas o desventajas.

2.1. ANÁLISIS Y EDUCCIÓN DE REQUISITOS

A continuación, se van a exponer los requisitos, identificados y consensuados con el cliente, tras las reuniones realizadas durante la primera fase del proyecto. Se han distinguido dos tipos de requisitos, funcionales y no funcionales. Los primeros se han organizado por características y los segundos por tipos.

2.1.1. DIAGRAMA DE CASOS DE USO

La **Figura 1** muestra el diagrama de casos de uso (a modo general) de la herramienta para tener, de manera rápida y concisa, una primera impresión sobre la funcionalidad requerida por la herramienta a implantar. Las **Figuras 2, 3 y 4** muestran el detalle de cada caso de uso representado en la figura anterior. La representación del diagrama se ha fragmentado debido al tamaño total del mismo.

Como podemos observar, el número de casos de uso que recoge el diagrama resulta bastante mayor que los realizados en las prácticas realizadas a lo largo de la carrera, dándonos una visión general de la cantidad de funcionalidad que abarca la herramienta desarrollada.

Para simplificar la representación, los enlaces entre los casos de uso y los actores que pueden realizarlos se han sustituido por un código de colores. Según este código, los casos de uso cuyo borde está coloreado de naranja son aquellos que el usuario puede realizar.

En el apartado 2.1.2 se describe de forma breve el detalle de los casos de uso más significativos incluidos en los dos primeros incrementos.

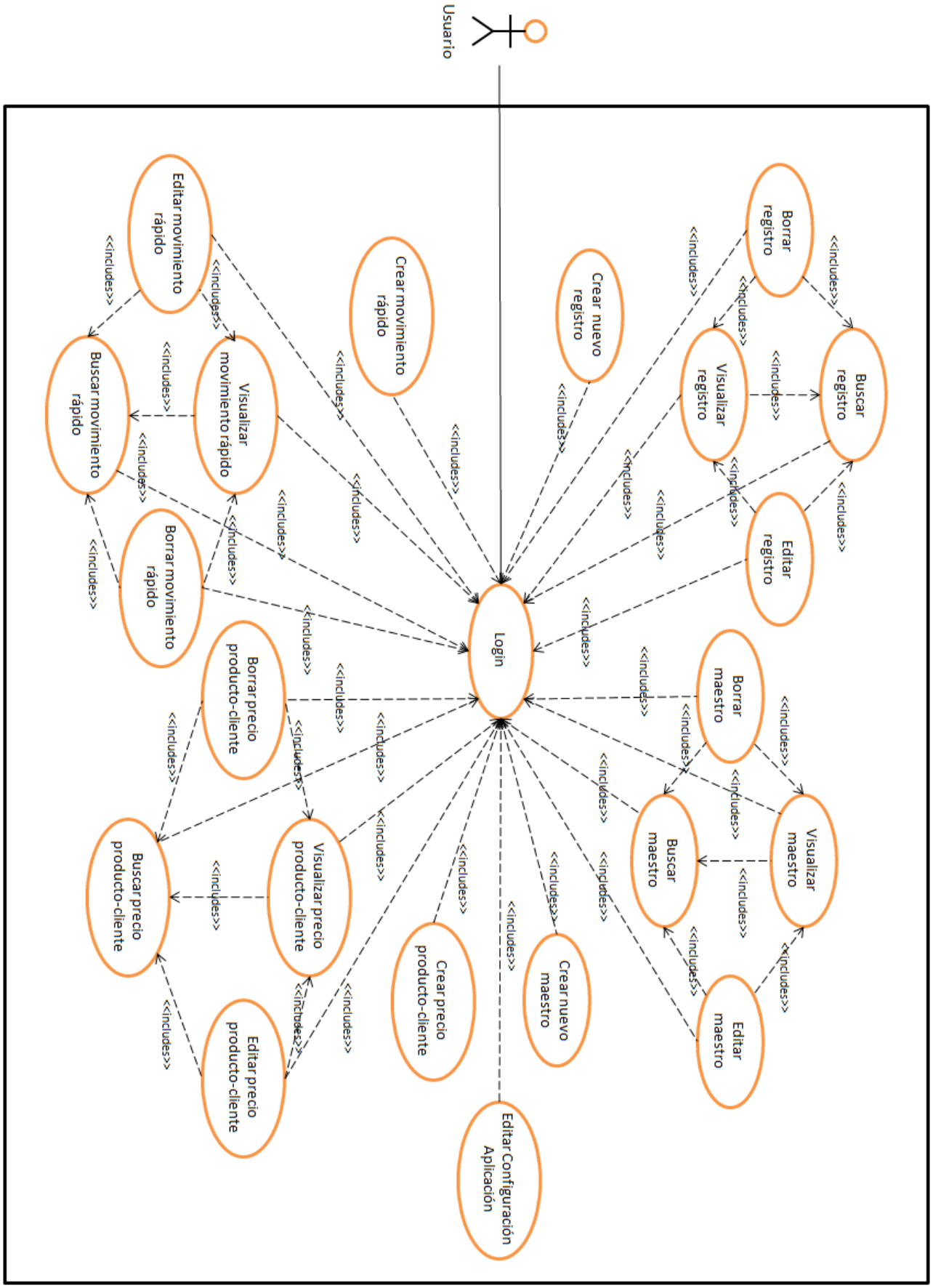

FIGURA 1. *DIAGRAMA DE CASOS DE USO (VISIÓN GENERAL).*

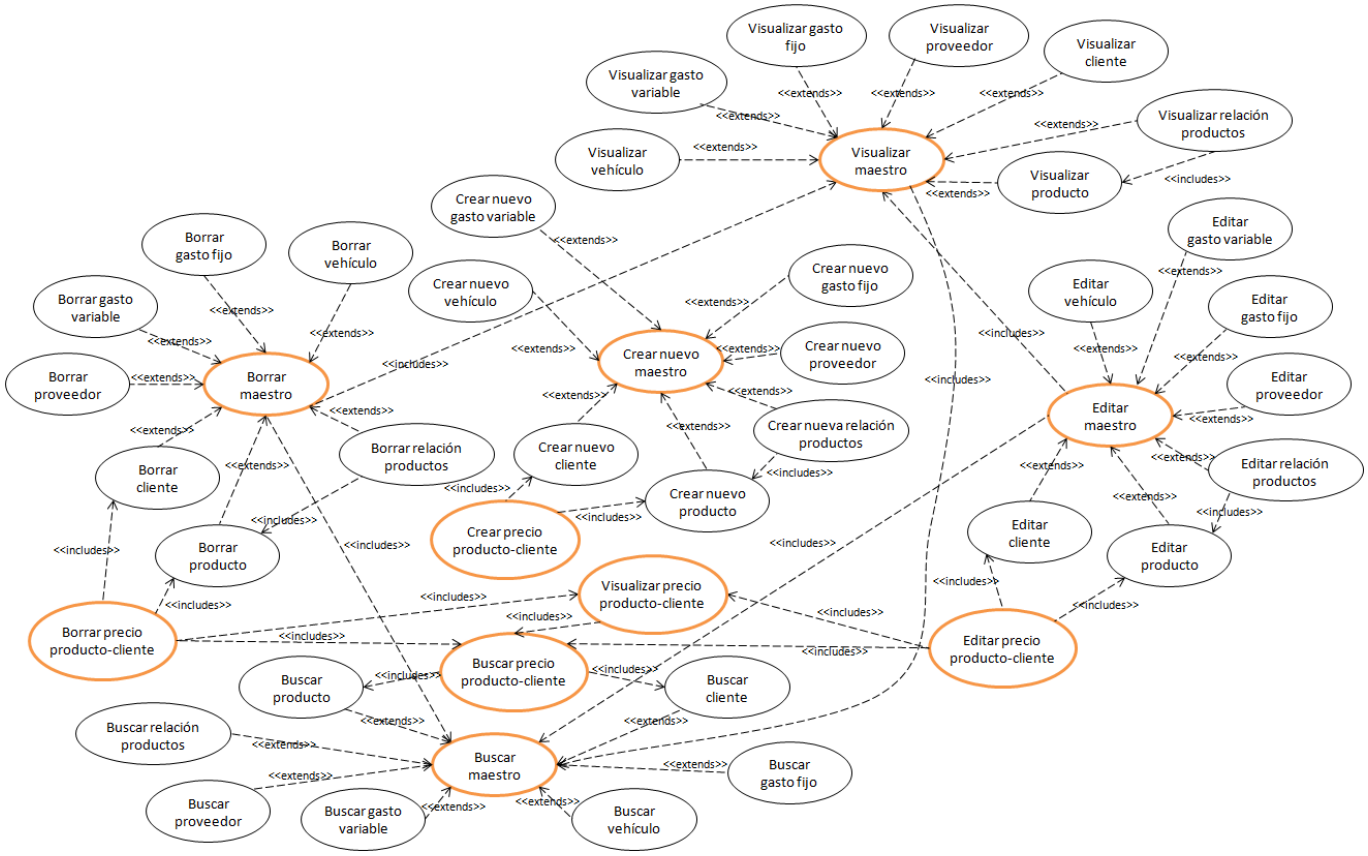

FIGURA 2. *CASOS DE USO DE MAESTROS Y RELACIÓN PRODUCTO-CLIENTE (DETALLE).*

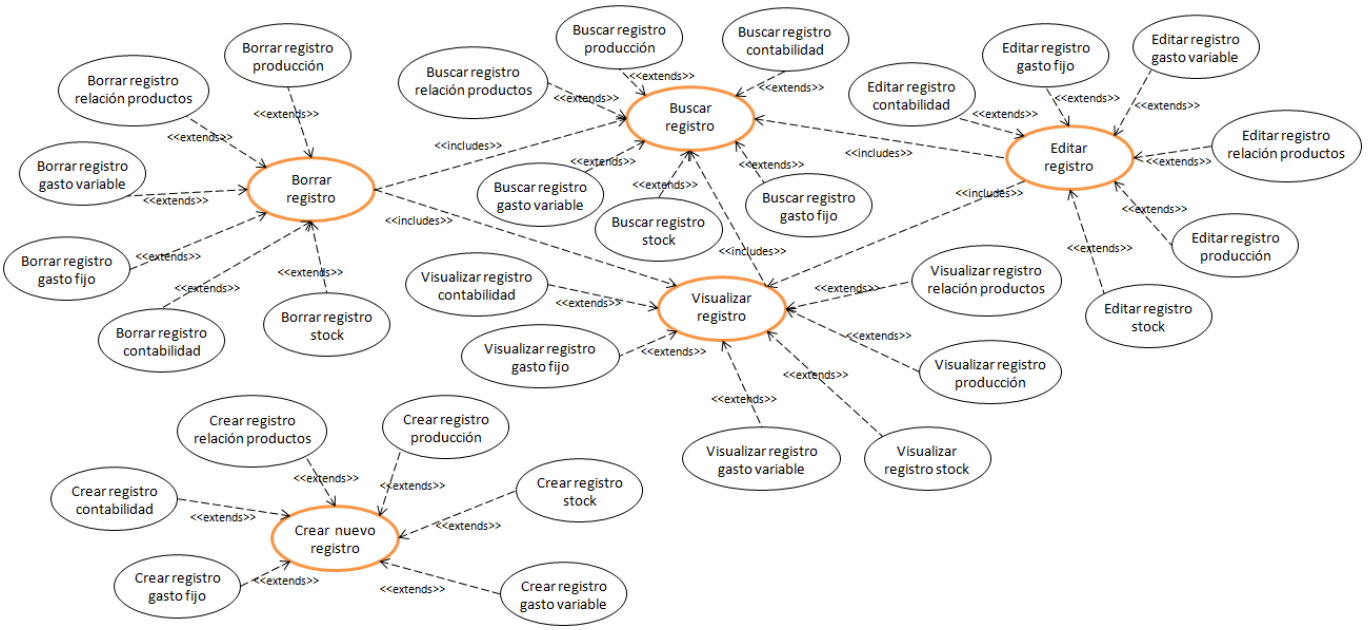

FIGURA 3. *CASOS DE USO DE REGISTROS (DETALLE).*

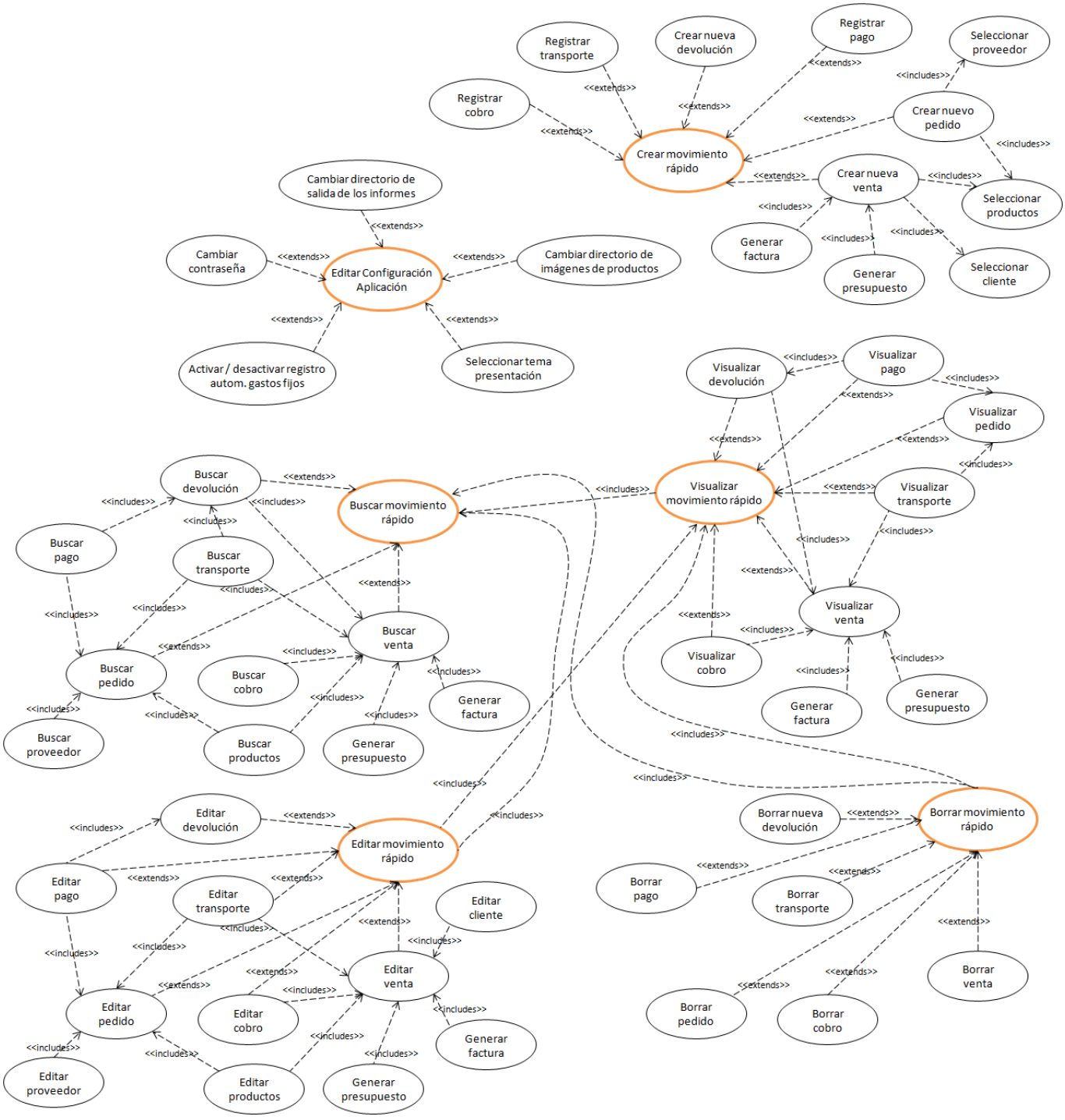

FIGURA 4. *CASOS DE USO DE MOVIMIENTOS RÁPIDOS Y CONFIGURACIÓN (DETALLE).*

2.1.2. REQUISITOS FUNCIONALES

INICIO

- **Introducción/Propósito de la característica**

  Esta característica se basa en permitir el acceso a la aplicación sólo al usuario para el cual se ha diseñado la herramienta. También contempla realizar las operaciones en base de datos que permiten actualizar, de manera automática, los registros de las tablas de histórico de stock e histórico de contabilidad.

- **Secuencia de estímulos/respuestas**

  El usuario abre la aplicación, se muestra una pantalla inicial con un diálogo de espera que se mantendrá mientras se realicen las operaciones de inicialización en la base de datos. A continuación, se solicitará autentificación para poder acceder a la herramienta. Una vez registrado, se accederá al menú principal de la aplicación que nos dará acceso a las diferentes funcionalidades del sistema.

- **Requisitos Funcionales asociados**

RF1. Operaciones de inicialización.

Realización de manera automática, al iniciar la aplicación, de la inserción en base de datos de registros pertinentes tales como los registros diarios de contabilidad y stock que faltasen por insertar hasta la fecha actual y la comprobación de los gastos fijos calendarizados para insertar, cuando corresponda, el registro del pago de los mismos en base de datos.

RF2. Autentificación.

El usuario se tendrá que autentificar para poder acceder a la aplicación. Para ello, será necesario que introduzca la contraseña codificada y almacenada en base de datos, la cual se establecerá en la puesta en producción del producto y se podrá modificar posteriormente desde el menú de configuración.

GESTIÓN DE DATOS

- **Introducción/Propósito de la característica**

  Esta característica se basa en permitir el acceso a los datos almacenados en la aplicación así como la posibilidad de realizar operaciones sobre los mismos, como son, insertar nuevos datos, consultar los datos existentes, actualizar y borrar.

- **Secuencia de estímulos/respuestas**

Desde el menú inicial, el usuario podrá acceder a la gestión de datos pinchando en el botón con tal nombre. Llegados a este punto, se mostrará un panel con botones grandes e ilustrados que permitan acceder a cada una de las entidades que se contemplan en la aplicación. Tales entidades serán: **clientes, productos, proveedores, vehículos, contabilidad, ventas, pedidos, stock, gastos fijos, gastos variables y producción**.

Una vez seleccionada alguna de las entidades, se mostrará un formulario que permitirá, en una misma ventana, realizar la inserción de nuevos registros, realizar búsquedas sobre los datos almacenados, los cuales se podrán filtrar por diferentes campos como códigos, nombres o rangos de fechas (dependiendo del formulario en cuestión), actualizar los datos almacenados y borrar los registros que se deseen.

- **Requisitos Funcionales asociados**

RF3. Insertar nuevo registro

El usuario deberá introducir todos los datos necesarios para realizar el registro completo. Tales datos, se validarán, mostrando el error específico producido en caso de haberlo, y en caso de pasar la validación, se procederá a insertar el nuevo registro en base de datos.

RF4. Buscar y filtrar registros

Se podrán seleccionar valores para los filtros que se deseen emplear a lo hora de buscar y mostrar los resultados obtenidos. Tales filtros permitirán la selección de valores de entre los posibles de una lista (ofreciendo además funcionalidad de autocompletado). En el caso de las fechas, se facilitará al usuario la elección de la misma ofreciéndole la posibilidad de escribirla o seleccionar mediante un selector de fechas. El usuario tendrá la opción de buscar utilizando uno o todos los filtros disponibles y seleccionados en el formulario.

RF5. Actualizar datos de registros

Sobre la tabla de registros que contenga los resultados de las búsquedas realizadas, se podrán realizar cambios de los datos que contengan. El usuario deberá modificar los datos que desee cambiar de un mismo registro y acto seguido pinchar en el botón de guardar cambios. Para evitar problemas de no saber qué datos se han modificado antes de guardarlos, sólo se actualizarán

los datos del último registro editado. Antes de guardar, se solicitará confirmación por parte del usuario.

RF6. Borrar un registro

Sobre la tabla de registros que contenga los resultados de las búsquedas realizadas, se podrá seleccionar un registro y borrarlo pinchando en el botón correspondiente. Antes de borrar, se solicitará confirmación por parte del usuario.

MOVIMIENTOS RÁPIDOS

- **Introducción/Propósito de la característica**

Dar acceso a las actividades más comunes y que requieren de una mayor rapidez de ejecución es lo que se trata en esta característica. Tales actividades son el registro y creación de una nueva venta, de una devolución, de un pago asociado a una venta o devolución, y de transportes que no se hayan registrado en el momento de creación o registro de las ventas o devoluciones.

- **Secuencia de estímulos/respuestas**

En el menú inicial hay un botón de acceso a los movimientos rápidos. Una vez seleccionado el movimiento rápido que se desee realizar, accederemos a una pantalla diseñada a medida para la actividad en cuestión a realizar.

En el caso de las ventas y los pedidos se emularán las interfaces empleadas típicamente en los sistemas TPV de los comercios, compuestas por 3 secciones diferenciadas: un panel dedicado a los datos del cliente o proveedor en cuestión (dependiendo de si se trata de una venta o un pedido), otro panel que recoja el "carrito de la compra" con los pedidos que se vayan seleccionando y un tercer panel, el de mayor extensión en la interfaz, que permita seleccionar los productos registrados en la aplicación agrupados por categorías; los productos se representarán mediante un botón, ilustrado con una imagen del producto o un cuadrado vacío en su defecto, de un tamaño que facilite su selección, y acompañado por un título descriptivo del producto en cuestión. Al seleccionar un producto, en caso de que haya varios modos, se preguntará el tipo de envasado que se desea.

Tanto para las ventas como para los pedidos, se podrá generar un presupuesto (simular la factura que resultaría del pedido o venta realizado) o confirmar la operación, registrando en este caso los datos en la aplicación y dando paso a la ventana de introducción del pago que se va a realizar en ese mismo momento (puede no pagarse nada, realizarse un pago parcial o un pago completo). Se ofrecerá también la posibilidad de registrar el transporte que suponga tal venta o

pedido (podrá ser un servicio propio mediante los vehículos de la empresa o un transporte contratado a un tercero, que se consideraría como un gasto variable a los ojos de la aplicación).

- **Requisitos Funcionales asociados**

RF7. Registrar nueva venta / pedido

En primer lugar, el usuario deberá seleccionar un cliente / proveedor registrado en la aplicación o introducir unos pequeños datos representativos del mismo si no se desea almacenarlo en la aplicación (también se ofertará la opción de acceder al formulario pertinente para poder registrar uno nuevo). A continuación se comenzarán a seleccionar los productos que se deseen incluir en la operación (en el caso de las ventas, si hay un registro guardado en la tabla de asociación cliente – producto, se empleará el precio que esté ahí almacenado, en caso de no haberlo, se utilizará el precio por defecto del producto). Una vez seleccionados los productos, los cuales aparecerán en el carrito de la compra, se podrán modificar las unidades, el precio por unidad y los descuentos que se deseen realizar. A continuación, se permitirá confirmar la operación, e introducir en la nueva ventana que aparecerá, el importe del pago a realizar así como indicar, en caso de realizarse, los datos del transporte asociado a tal operación. Cuando se confirme una venta, el cliente tendrá la opción de imprimir una factura (Incremento II). Si no se desea confirmar la operación, se ofrece la posibilidad de generar un presupuesto para la venta realizada (Incremento II).

RF8. Registrar nuevo pago asociado a una venta / pedido

Se podrá registrar un nuevo pago asociado a una venta o pedido que estén todavía pendientes de pago.

RF9. Registrar nuevo transporte asociado a una venta / pedido

Se podrá registrar un nuevo transporte asociado a una venta o pedido que haya sido registrado previamente. Actualmente se registrará, en el caso de realizarse mediante los vehículos de la empresa, el vehículo utilizado, la fecha, el precio cobrado al cliente (en el caso de las ventas) y una aproximación de los kilómetros realizados (queda como futura mejora o ampliación la posibilidad de utilizar el API de Google para el cálculo de la ruta mediante la introducción de la dirección destino, la cual se podrá extraer automáticamente del cliente en caso de que esté registrado en la aplicación).

RF10. Registrar nueva devolución asociada a una venta

Se podrá registrar una devolución, por parte de un cliente, de los adquiridos en una venta en particular. Se deberá indicar la venta asociada a la devolución, así como los productos que se deseen devolver de los adquiridos previamente.

CONFIGURACIÓN

- **Introducción/Propósito de la característica**

En esta característica se ofrece la posibilidad de cambiar ciertos parámetros relacionados con los aspectos de configuración de la aplicación. Tales aspectos son, actualmente, la posibilidad de cambiar la contraseña de acceso a la aplicación, así como elegir si se desea que en las operaciones de inicialización de la aplicación se inserten automáticamente los registros derivados de los gastos fijos pertinentes. Se plantean como futuras opciones de configuración la elección del tema o estilo de la interfaz, cambiar los directorios por defecto donde se depositarán los informes generados por la aplicación o donde se almacenarán las imágenes de los productos utilizadas en las ventanas de creación de nuevas ventas y pedidos.

- **Secuencia de estímulos/respuestas**

Se accederá a la configuración a través del botón destinado a ello del menú inicial. La ventana será un formulario típico de configuración, donde se podrán marcar o desmarcar las opciones ofrecidas, así como introducir texto (en el caso de solicitar un cambio de contraseña) o seleccionar un nuevo directorio destino para informes en el caso de las futuras mejoras.

- **Requisitos Funcionales asociados**

RF11. Cambio de contraseña

En primer lugar, el usuario deberá introducir la contraseña actual, la cual, una vez verificada, se sustituirá por la nueva contraseña que introduzca el usuario, que deberá introducirse por duplicado para evitar posibles errores tipográficos que ocasionasen un problema en los futuros accesos a la aplicación.

RF12. Activar / desactivar los registros automáticos de gastos fijos

El usuario podrá decidir si desea que se realicen los registros de forma automática al iniciar la aplicación o si prefiere hacerlo él manualmente los días que correspondan.

RF13. Cambiar el directorio de salida de informes (Incremento II)

El usuario podrá seleccionar o escribir el directorio en el cual desea que se depositen por defecto los informes que genere la aplicación.

RF14. Cambiar el directorio de las imágenes de los productos (Incremento II)

El usuario podrá seleccionar o escribir el directorio en el cual desea que se busquen las imágenes que se utilizarán para pintar los productos en las ventanas de creación de nuevas ventas o pedidos.

RF15. Cambiar el tema de la aplicación

El usuario podrá seleccionar, de entre una lista predefinida, el tema que desee que tenga la interfaz de la aplicación.

INFORMES (INCREMENTO II)

- **Introducción/Propósito de la característica**

Con esta característica queremos conseguir que la aplicación sea capaz de calcular y generar informes de la empresa que resulten útiles al usuario para poder analizar el avance y estado de la empresa, ya sea mediante informes diarios, mensuales, trimestrales o anuales. Tales informes se calcularán a partir de los datos que la aplicación va almacenando y recogiendo durante el día a día de la empresa.

- **Secuencia de estímulos/respuestas**

Los informes se podrán visualizar en la aplicación accediendo a la sección de informes desde el menú inicial. Podrán calcularse para un rango de fechas determinado y se podrán visualizar desde la propia aplicación o generando un fichero en el directorio de salida determinado en la configuración de la aplicación.

- **Requisitos Funcionales asociados**

RF16. Generar Balance y cuenta de Pérdidas y Ganancias.

Atendiendo a las cuentas del Plan General Contable [Ref. (1)] que en la empresa analizada se utilizan, el usuario podrá generar un informe de la cuenta de Pérdidas y Ganancias así como del Balance de la empresa.

RF17. Generar informe de rendimientos de productos

El usuario podrá seleccionar los productos para los cuales desee generar un informe de rendimiento.

RF18. Generar informe de rendimientos de clientes

El usuario podrá seleccionar los clientes para los cuales desee generar un informe de rendimiento.

2.1.3. REQUISITOS NO FUNCIONALES

SEGURIDAD

RNF1. Privacidad y seguridad de los datos

Para poder trabajar con la aplicación es necesario hacer login para garantizar que ninguna persona ajena a la gestión de la empresa pueda tener acceso a los datos, garantizando así la integridad y privacidad de los mismos.

MANTENIMIENTO Y PORTABILIDAD

RNF2. Mantenimiento

La aplicación deberá ser flexible para permitir la introducción de mejoras y cambios en la herramienta con el menor coste posible, ahorrando en tiempo y dinero.

RNF3. Portabilidad

En un futuro, debe ser portable a dispositivos Android.

DE RECURSOS

RNF4. Volúmenes de datos.

No hay límite de productos, clientes, proveedores ni cualquiera del resto de entidades con las que trabaja la herramienta. Sólo se debe tener en cuenta que, al tener tablas de registro, cada cierto tiempo se ha de realizar un mantenimiento de las mismas, conservando en las tablas que más a menudo utiliza la aplicación una cantidad de registros (asociado a horizontes temporales definidos, como por ejemplo, una ventana de un año) razonable y que no ralentice los accesos a base de datos por parte de la aplicación, y sacando a tablas de histórico el resto de datos recopilados.

DE USABILIDAD

RNF5. Facilidad de navegación.

Es primordial que la organización de las actividades en menús y la navegación tenga un sentido lógico que favorezca el proceso de adaptación y formación del usuario.

RNF6. Introducción de datos.

La introducción de datos deberá estructurase de una manera sencilla, procurando minimizar los errores (desplegar un menú en lugar de solicitar por teclado una opción) cometidos por parte del usuario. También debe ser rápida a la hora de introducir datos en operaciones críticas para la actividad diaria del usuario, como es la introducción de nuevas ventas, siendo un requisito poder registrar una venta en menos de un minuto.

DE IMPLEMENTACIÓN

RNF7. Lenguaje de programación de la aplicación.

La aplicación se implementará en lenguaje Java, ya que al plantearse como futura mejora adaptar la herramienta a dispositivos con Android, facilitará mucho el trabajo a realizar para llevar a cabo la portabilidad.

RNF8. API de Google Maps (Incremento II).

Para un futuro cercano se incluirá en la herramienta el API de Google para el cálculo de rutas que facilite el registro de los datos originados por los transportes, así como la eficiencia y rapidez de ejecución de los mismos.

2.2. DESCRIPCIÓN HERRAMIENTAS EXISTENTES

Como ya se ha mencionado en puntos anteriores del documento, actualmente hay un gigantesco mercado de aplicaciones de gestión empresarial. La mayoría de las herramientas con precios asequibles para autónomos y PYMEs son de ámbito genérico. Normalmente ofrecen funcionalidad muy extensa y definida para un tipo de necesidades, pero funcionalidad escasa o nula para otros tipos.

Existen herramientas para la emisión de facturas, contabilidad, compras, ventas, gestión de fabricación, inventario, RRHH y agenda, pero no herramientas que aúnen toda esta funcionalidad en una única aplicación. Si queremos una aplicación que nos ofrezca toda esta funcionalidad debemos recurrir a empresas de desarrollo de software a medida.

A continuación veremos algunos ejemplos de software de gestión empresarial (con módulos de funcionalidad independiente) y de empresas que nos ofertan ERP o BPMS a medida para las condiciones y necesidades de la empresa en cuestión.

2.2.1. HERRAMIENTAS DE GESTIÓN EMPRESARIAL

CONTALUX

Programa exclusivamente para **contabilidad,** difícil de utilizar y poco atractivo para el usuario. Son necesarios demasiados conocimientos en contabilidad para poder sacarle partido de verdad. Cuenta con multitud de ventanas muy pequeñas con la información muy concentrada y estresante para el usuario. Permite exportar información en Excel.

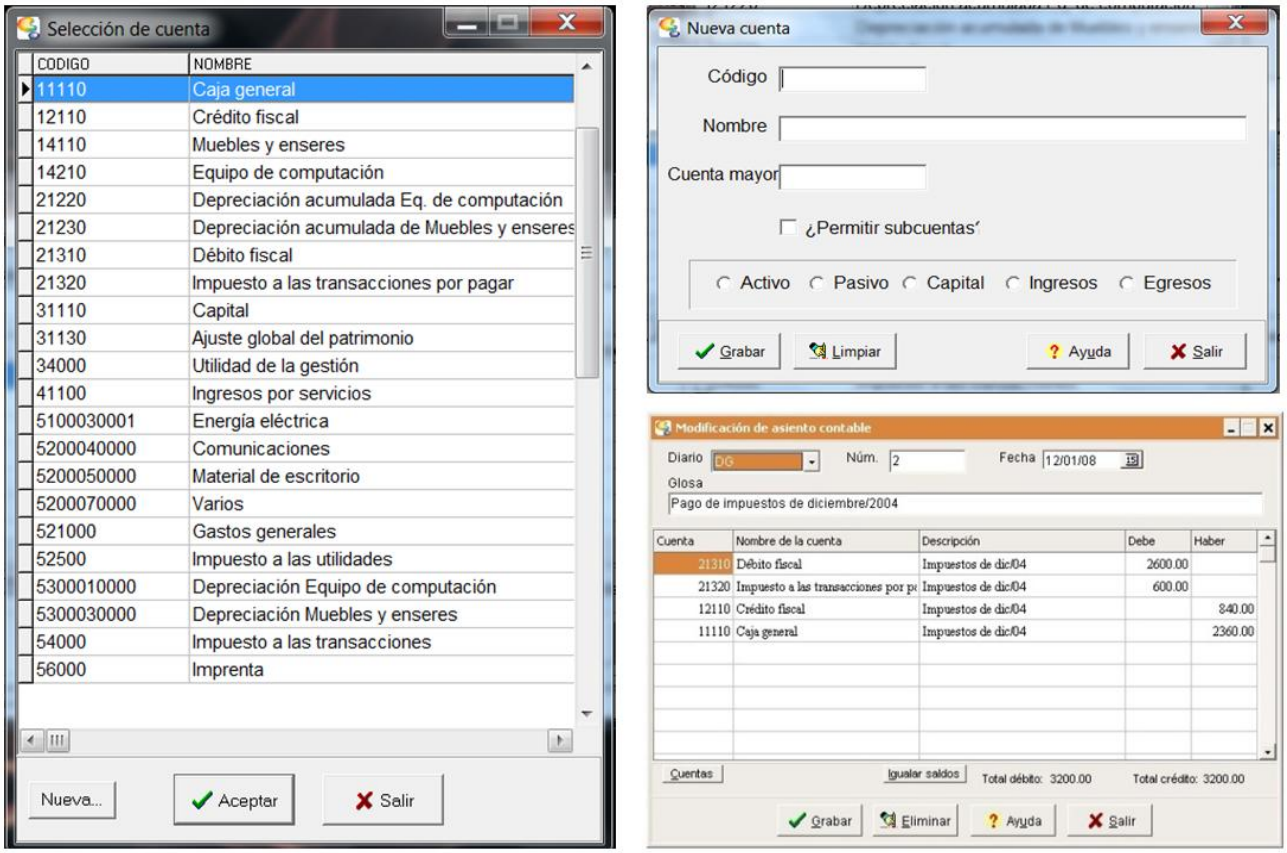

FIGURA 5. *CAPTURAS CONTALUX.*

SECOP

Software para llevar un completo control sobre **inventarios**. Puedes registrar ventas, compras, productos, añadir lista de precios y generar reporte de existencias.

Fácil de utilizar pero excesivamente simple y poco atractiva. No permite exportar información a Excel. Pertenece al mismo proveedor que la anterior herramienta (Contalux).

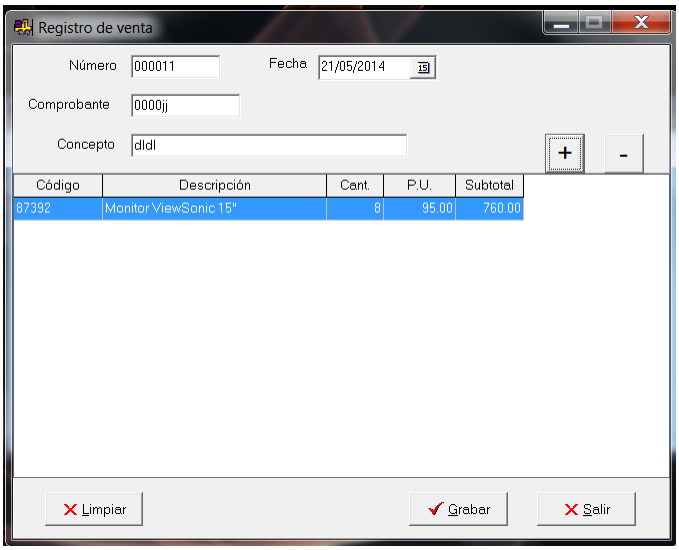

FIGURA 6. *CAPTURA SECOP.*

ALVENDI

Programa para gestionar **inventarios**. Su interfaz amigable lo convierte en una sencilla y práctica opción para registrar las compras o entradas de productos y las ventas o salidas.

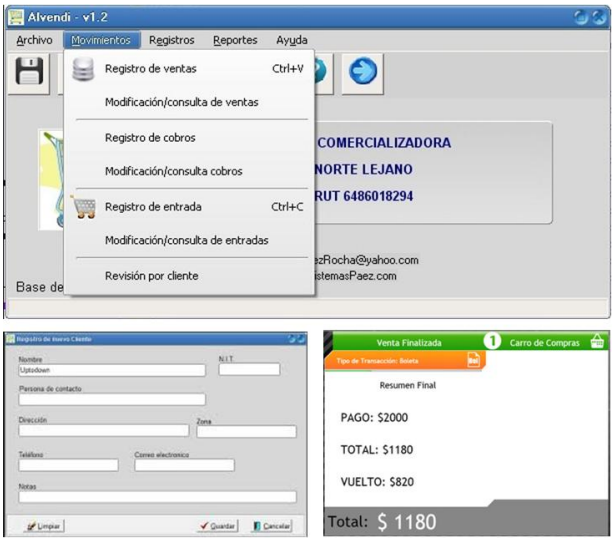

FIGURA 7. *CAPTURAS ALVENDI.*

SOFTWARE DEL SOL

Es un paquete de herramientas empresariales muy completo que incluye 8 herramientas diferentes: ContaSOL, FactuSOL, NominaSOL, TpvSOL, MovilSOL, InventaSOL, AgendaSOL y VisorSOL.

La interfaz recuerda mucho a la utilizada por las herramientas del paquete office de Microsoft.

Cada sistema cumple con las funciones que se esperan de ellas, pero tiene la principal desventaja de tener que instalar 8 aplicaciones diferentes para llevar toda la gestión de la empresa.

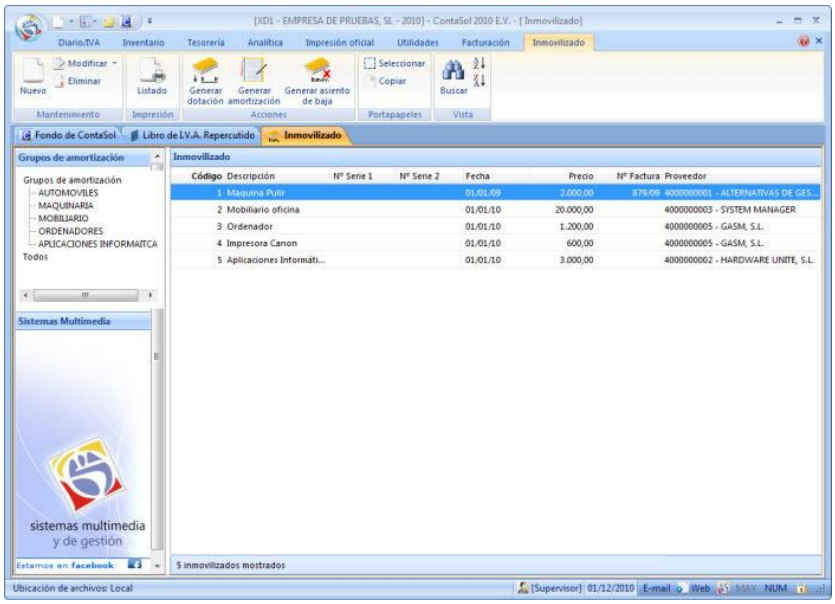

FIGURA 8. *CAPTURA CONTASOL.*

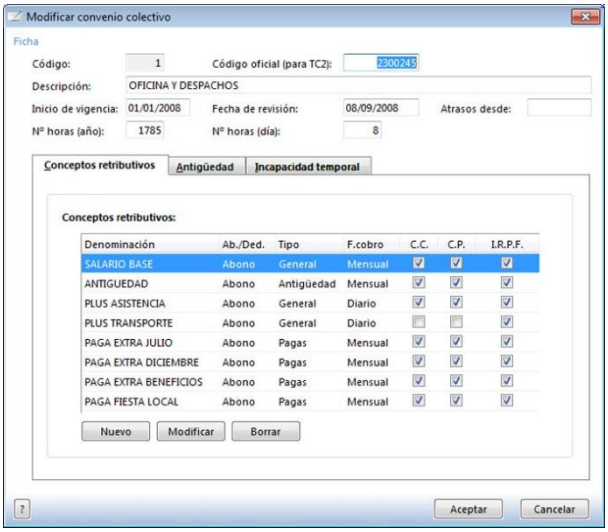

FIGURA 9. *CAPTURA NOMINASOL*

OPEN ERP (ODOO)

Es una de las mejores opciones que se puede encontrar en el mercado atendiendo a la relación calidad-precio. Se trata de un sistema software ERP (también incluye TPV y CRM) de código abierto que se plantea como alternativa a ***SAP ERP*** y ***Microsoft Dynamics***.

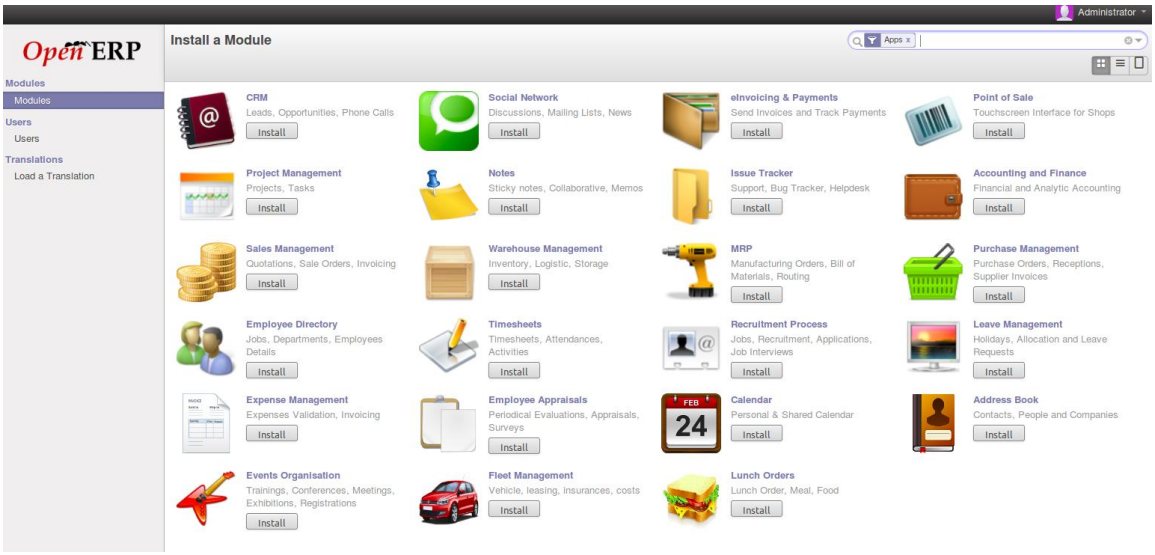

FIGURA 10. *CAPTURA 1 OPENERP (ODOO).*

Viene provisto de módulos estándar tales como, Gestión de compraventa, CRM, Gestión de proyectos, Sistema de gestión de almacenes, Manufactura, Contabilidad analítica y financiera, Puntos de venta, Gestión de activos, Gestión de recursos humanos, Gestión de inventario, Ayuda técnica, Campañas de marketing o Flujos de trabajo. Además, es una herramienta multiplataforma que cubre los sistemas operativos más comunes (Windows, Linux y Mac OS).

FIGURA 11. *CAPTURA 2 OPENERP (ODOO).*

Ofrecen tres opciones, una gratuita de funcionalidad más reducida, sin soporte, sin migración de datos y una de sus principales desventajas, es que no permite la creación de módulos privados. La segunda opción se trata de una solución online, también de funcionalidad reducida, que cuenta con un servicio de soporte y mantenimiento y tiene un precio de 39 € mensuales. La tercera, y más completa, opción sale por 165 € mensuales, y a parte de ofrecer todos los servicios estándar así como de mantenimiento y soporte, permite la utilización de módulos privados almacenados en el propio equipo del cliente.

Para configurarlo se requieren algunos conocimientos avanzados en el uso de equipos informáticos, y si se desea personalizar por completo, se debe programar la funcionalidad adicional para la cual no traiga soporte, aunque para las necesidades de una PYME, seguramente no sería necesario.

FIGURA 12. *CAPTURA 3 OPENERP (ODOO).*

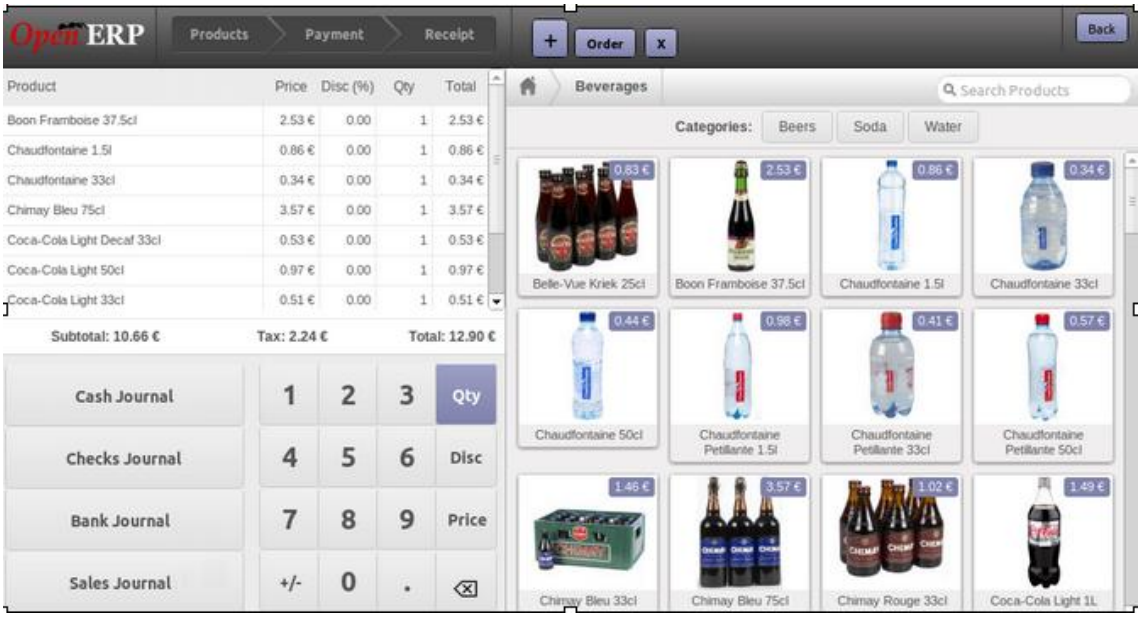

FIGURA 13. *CAPTURA 4 OPENERP (ODOO)*

FACTOOL V1.3.

Herramienta para registrar, imprimir y consultar **facturas emitidas**. Interfaz simple pero bastante intuitiva. Permite grabar productos en una lista, así que para las empresas que utilicen más o menos los mismos productos (como es nuestro caso) es bastante cómoda y permite ahorrar bastante tiempo. Uno de los fallos que tiene reside en que no permite exportar en Excel.

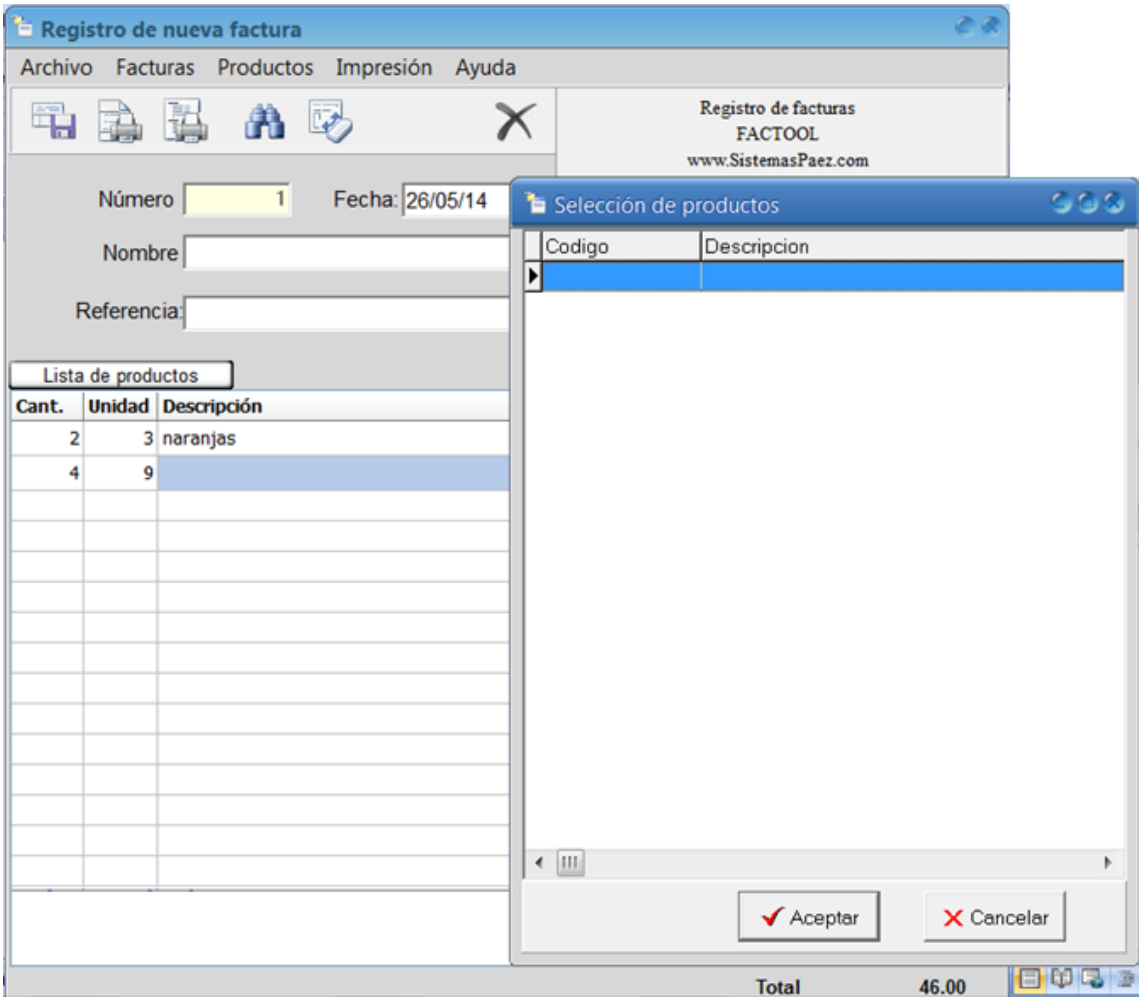

FIGURA 14. *CAPTURAS FACTOOL V1.3.*

JBPM Y JBOSS SOFTWARE

jBPM[1] es un motor que nos permite ejecutar procesos de negocio previamente introducidos mediante una descripción gráfica. Los procesos se componen de tareas que se conectan mediante flujos de secuencia (similares a los diagramas de transición de estados).

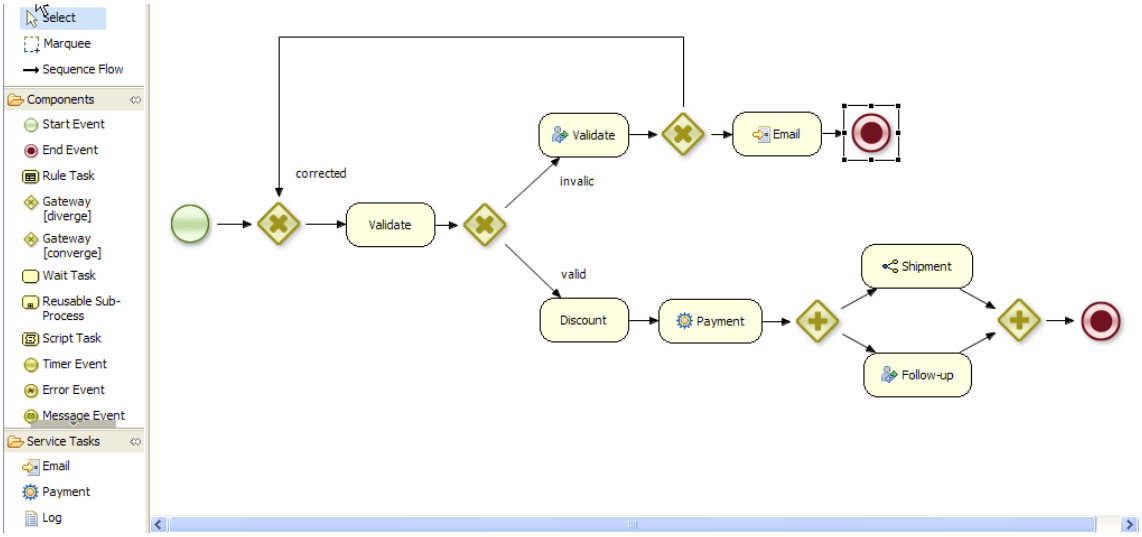

FIGURA 15. *EJEMPLO JBPM ECLIPSE PLUGIN.*

Las ejecuciones de un proceso se denominan instancias del proceso y jBPM las administra de manera que o bien se ejecutan operaciones automáticamente o se mantienen las instancias en estado de espera hasta que tenga lugar un evento (similar al funcionamiento de los autómatas).

Existen herramientas de la comunidad JBOSS[2] y plugins (por ejemplo, para eclipse) que permiten a los desarrolladores emplear jBPM. También existen editores web.

Requiere de conocimientos especializados para poder desarrollar una herramienta a medida.

[1] http://es.wikipedia.org/wiki/JBPM
[2] http://www.jboss.org/

2.2.2. OFERTA SOFTWARE ERP "A MEDIDA"

Debido a la falta de conocimiento acerca de la oferta de herramientas a medida, se ha utilizado un buscador[3] mediante el cual, indicando las características que deseamos tenga nuestra herramienta, nos devuelve un ranking de proveedores de software ERP que mejor se adaptan a nuestras necesidades. En la **Figura 16** se muestran las opciones seleccionadas en la búsqueda realizada.

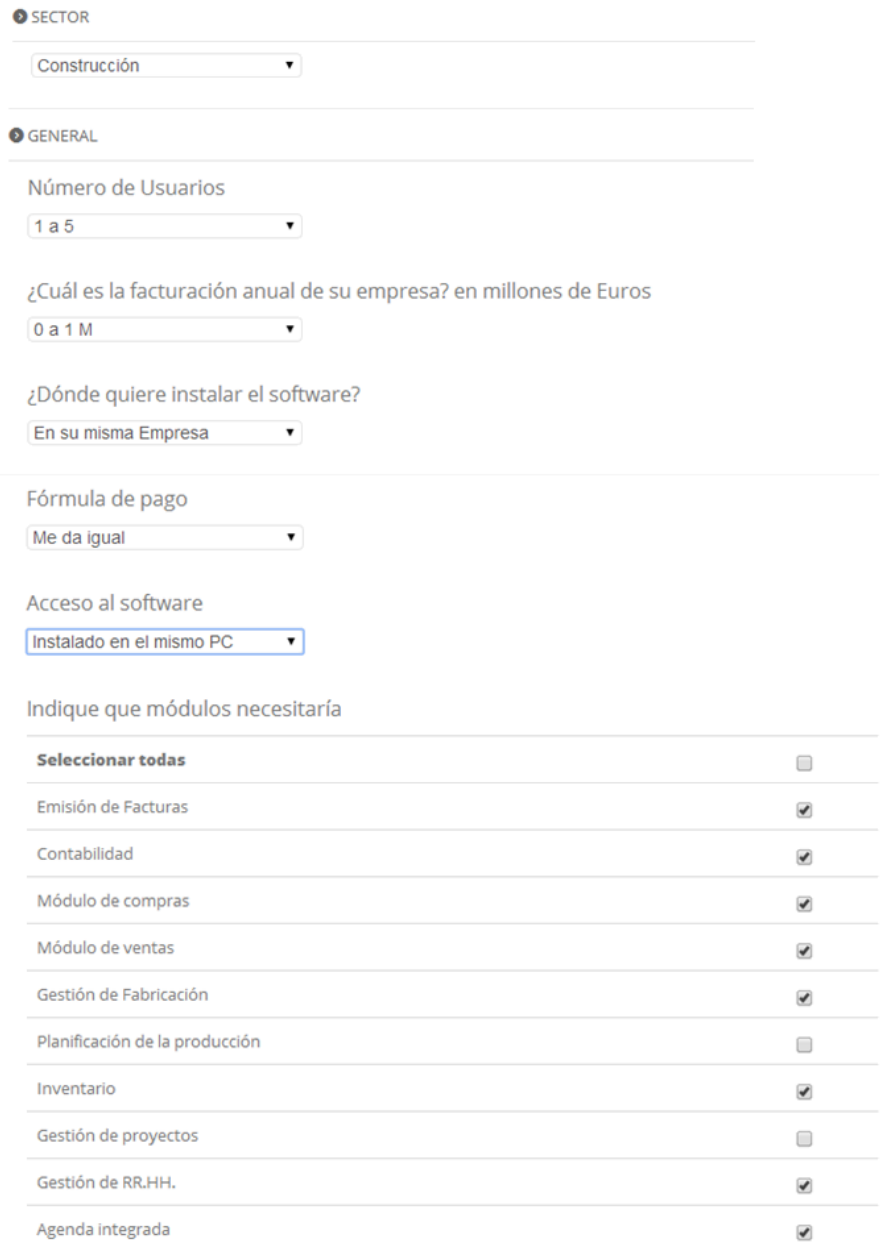

FIGURA 16. *FILTROS BÚSQUEDA EN BUSCOELMEJOR*

[3] http://www.buscoelmejor.com/mejor-erp/index.html

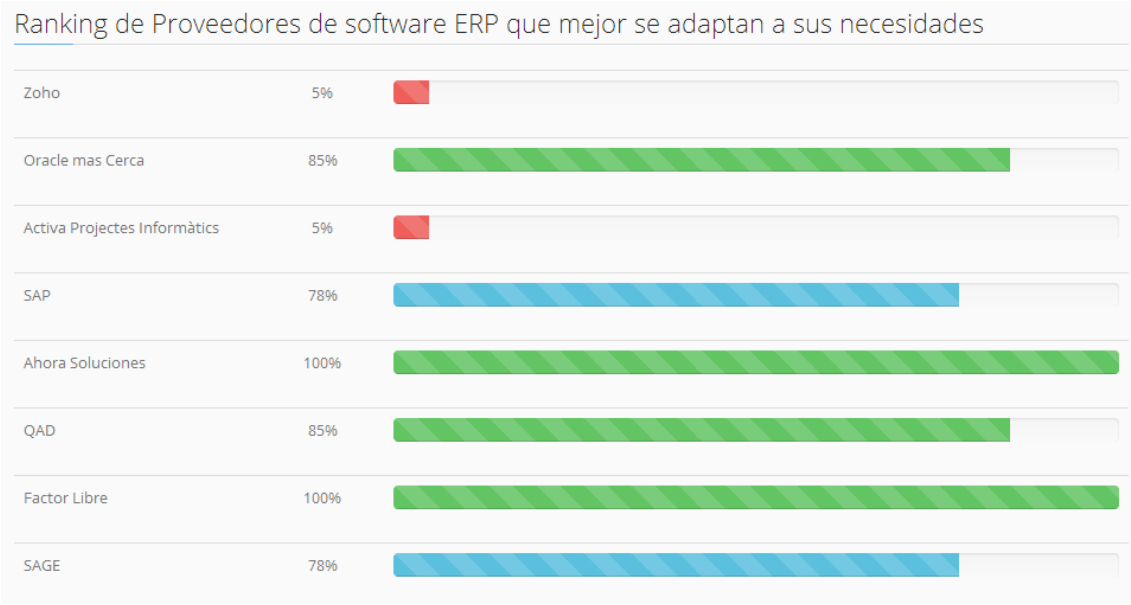

FIGURA 17. *RANKING DE PROVEEDORES (MUESTRA PARCIAL)*

Dentro del ranking de proveedores devuelto encontramos un total de 25 resultados, de los cuales, podemos destacar aquellos que tienen un porcentaje de adaptación superior o igual al 85%.

NOMBRE	PORCENTAJE DE ADAPTACCIÓN
Ahora Soluciones	100%
Factor Libre	100%
Primavera BSS	100%
Microsoft	100%
Exact	100%
Oracle más cerca	85%
QAD	85%
Wolters Kluwer | A3 Software	85%

TABLA 1. *MEJORES SOLUCIONES DEL RANKING.*

2.3. LIMITACIONES DE LAS HERRAMIENTAS EXISTENTES

Atendiendo a lo visto en los apartados 2.1 y 2.2, podemos hacer un resumen de las características de las herramientas que el mercado actual nos ofrece:

- Las aplicaciones de gestión modulares son mucho más baratas y rápidas de implantar pero más reducidas y difíciles de elegir.

- Las herramientas a medida son generalmente mejores, más completas, más caras y más lentas de implantar, además de incluir la desventaja de tener que, en la mayoría de los casos, contratar un mantenimiento de la aplicación con la misma empresa que nos la ha desarrollado.

En cuanto a precios, podemos diferenciar, de forma aproximada, tres rangos de precios orientados a diferentes tamaños de empresa; los dos primeros engloban software ERP a medida, y el tercero, soluciones comerciales o de software libre que se pueden descargar directamente de internet:

1. ***SAP*** o ***Microsoft Dynamics*** son algunas de las mayores que podemos encontrar y están destinadas principalmente a empresas grandes y de un alto capital. (100000 euros hacia arriba)

2. En el mercado de las PYMES podemos encontrar ***Microsoft Navision, SAGE X3, Solmicro Expertis, SAP Bussines one, A3 ERP (Wolters Kluwer)*** (40.000 – 80.000 euros)

3. Para soluciones pequeñas y medianas existen todo tipo de soluciones en internet que se pueden descargar directamente para su uso. Algunas tienen un coste fijo, otras se alquilan y algunas son soluciones de Software Libre, como las mencionadas en el apartado 2.2 (30.000 euros hacia abajo)

Se debe tener en cuenta que los precios indicados son una aproximación, dado que en el precio de este tipo de herramientas influyen mucho las características exactas del sistema a desarrollar, la empresa que lo contrate y la empresa que lo desarrolle.

2.4. APORTACIONES DE LA HERRAMIENTA DESARROLLADA

La herramienta desarrollada, cumpliendo con los requisitos acordados con el cliente (apartado 2.1) y atendiendo a las ventajas y desventajas de la herramientas analizadas anteriormente (apartados 2.2 y 2.3), se podría describir como una herramienta con la personalización y funcionalidad, en menor escala, ofrecida por un software ERP a medida con los costes de una herramienta de gestión empresarial de ámbito comercial (en nuestro caso, coste 0).

Las características a destacar de la herramienta son:

- Aúna en una única aplicación la funcionalidad que otros sistemas abarcan en 7 u 8 herramientas diferentes, como son la gestión de clientes, productos, proveedores, vehículos, contabilidad, ventas, facturación, pedidos, stock, gastos fijos, gastos variables y producción.

- Cuenta con un estilo, a nivel estético y conceptual, consensuado y hecho a medida para el cliente en cuestión.

- La navegación entre menús y secciones de la aplicación resulta sencilla e intuitiva.

- Al contar con un motor de base de datos, la persistencia de los mismos, así como la facilidad de realizar backups, está garantizada.

- Supone un salto muy importante para la gestión empresarial del cliente en cuestión, dejando atrás una gestión anticuada, basada *"en el papel y el boli"*, y adaptándose a las ventajas que la tecnología actual nos ofrece, suponiendo a corto y largo plazo, un ahorro en tiempo y dinero.

- Está diseñada para crecer y adaptarse a las necesidades del cliente.

2.5. CICLO DE VIDA SOFTWARE

Para analizar qué MCVS (Modelo de Ciclo de Vida Software) es el que mejor se adapta al proyecto, vamos a estudiar punto a punto diferentes criterios estructurados en 5 aspectos globales distintos [Ref. (2)].

2.5.1. VALORES DE LOS CRITERIOS DEL PERSONAL

C1. Experiencia del usuario en el ámbito de la aplicación

El cliente se enfrenta por primera vez al desarrollo y contratación de este tipo de servicio, por lo que su experiencia se corresponde con un nivel de principiante, siendo menos restrictivo con el producto deseado por falta de experiencia, y a la vez, requiriendo una aplicación que sea fácil de utilizar. *(Criticidad alta).*

C2. Capacidad del usuario para expresar los requerimientos del sistema

Por su falta de experiencia, el cliente tiene dificultades para expresar cómo quiere exactamente que sea la aplicación, pero es capaz de reconocer aquello que le gusta cuando lo ve. *(Criticidad media).*

C3. Experiencia del desarrollador en el ámbito de la aplicación

La experiencia y conocimientos adquiridos a lo largo de la carrera universitaria, incluyendo los 10 meses trabajados en el área de desarrollo de una consultoría económica, son suficientes y adecuados para el desarrollo de la aplicación. *(Criticidad media).*

C4. Experiencia del desarrollador con el entorno de trabajo

Los diferentes entornos de trabajo en los que se desarrollará la aplicación son conocidos y se lleva trabajando con ellos desde hace 4 años. *(Criticidad baja).*

2.5.2. VALORES DE LOS CRITERIOS DEL PROBLEMA

C5. Madurez de la aplicación

El problema que abarca esta herramienta se encuentra en un estado de madurez sólido y bien consolidado, existiendo numerosas herramientas y bases definidas que afrontan problemas de este ámbito. Sin embargo, el avanzado estado del arte no implica un menor esfuerzo para la consecución de los objetivos. *(Criticidad media).*

C6. Complejidad del problema

Este tipo de aplicaciones es un tema bastante extendido y utilizado pero el hecho de ser un único trabajador y el poco tiempo disponible para desarrollar el proyecto, dificultan la labor. *(Criticidad media).*

C7. Requerimientos de funcionalidad parcial

Este tipo de aplicación permite que se desarrollen funcionalidades parciales de la misma, ya que cuenta con diferentes módulos y funcionalidades independientes entre sí, aunque se dan algunos casos en los que los módulos deben estar interconectados. Además, puesto que el cliente no está familiarizado con el ámbito de la aplicación y tiene dificultades para expresar sus requerimientos, es recomendable realizar entregas parciales. *(Criticidad alta).*

C8. Frecuencia de cambios

Por sí mismo el problema no generaría cambios en los requerimientos o diseño de la aplicación, pero debido a la inexperiencia del cliente, es posible que cambie de opinión a medida que se avance en el desarrollo de la aplicación. *(Criticidad media).*

C9. Magnitud de los cambios

Los cambios que se produzcan no deberían modificar en exceso el diseño y desarrollo de la aplicación, aunque no es una medida exacta y depende de las solicitudes del cliente. *(Criticidad media).*

2.5.3. VALORES DE LOS CRITERIOS DEL PRODUCTO

C10. Dimensión del producto

Un sistema como el solicitado por el cliente, que conlleva una alta carga de trabajo en el ámbito funcional, será un sistema de dimensión media-alta. *(Criticidad media).*

C11. Complejidad del producto

La dificultad de desarrollo del producto se puede considerar como alta, y más teniendo en cuenta la poca disponibilidad de recursos humanos para llevarlo a cabo. *(Criticidad alta).*

C12. Requerimientos cualitativos

Se requiere que el producto sea usable, fiable y de fácil mantenimiento, probablemente, un poco más que otras aplicaciones. *(Criticidad media).*

C13. Requerimientos de interfaz de usuario

La interacción del usuario con la aplicación es de gran importancia, ya que será determinante para el mismo si el aspecto de la aplicación es agradable, si es fácil de utilizar y recordar, etc. *(Criticidad alta).*

2.5.4. VALORES DE LOS CRITERIOS DEL RECURSO

C14. Financiación

En el proyecto que nos compete, este criterio no aplica debido a que no existe financiación. No se precisan licencias y todas las herramientas empleadas son gratuitas. En el caso de tratarse de un producto para cualquier otro cliente, en el **Anexo I** se puede consultar un ejemplo de presupuesto válido para el proyecto. *(Criticidad media).*

C15. Disponibilidad de fondos

Para un proyecto de este estilo, la disponibilidad para emplear fondos en un esfuerzo no es elevada pero tampoco reducida, dado que una escasa disponibilidad de fondos supone una pérdida de calidad en el producto, por lo que debe estar equilibrada. *(Criticidad media).*

C16. Perfil del personal

Es suficiente una plantilla reducida para la realización del proyecto (en este caso una persona), siendo mayor la necesidad de personal al comienzo, debido a la fase de análisis del problema y de entendimiento con el cliente, cuya consecución inquirirá en un elevado número de reuniones. *(Criticidad media).*

C17. Disponibilidad de personal

De nuevo, al tratarse de un TFG, este criterio tampoco aplica. En condiciones normales, no sería necesaria una disponibilidad de 24 h por parte de los trabajadores al no tratarse de una aplicación que requiera un mantenimiento de emergencia. *(Criticidad baja).*

C18. Accesibilidad a los usuarios

Se tendrá acceso al usuario final de la aplicación de manera frecuente aunque sin abusar de ello. En concreto, se estima mantener una reunión por semana. *(Criticidad alta).*

2.5.5. VALORES DE LOS CRITERIOS ORGANIZACIONALES

C19. Gestión de compatibilidad

Los requerimientos organizativos de desarrollo se adecúan sin grandes complicaciones al modelo de proceso software. *(Criticidad media).*

C20. Garantía de calidad y metodología administrativa

Del mismo modo que en criterios anteriores, el hecho de tratarse de un TFG implica que no se precisen certificados de calidad y metodología administrativa que abalen a la empresa desarrolladora, por lo que este criterio no aplica. *(Criticidad baja).*

2.5.6. JUSTIFICACIÓN DE LA SELECCIÓN DEL MODELO DE CICLO DE VIDA SOFTWARE

De acuerdo con los puntos estudiados y mediante el algoritmo de selección por tablas [Ref. (2)] (**Tabla 2** y **Tabla 3**) empleado en la asignatura *Ingeniería del Software*, llegamos a la conclusión de que los modelos de proceso más adecuados de nivel 3 y 2 para el proyecto son el *modelo incremental* [Ref. (3)] (nivel 3) y el *modelo híbrido basado en prototipos* [Ref. (2)] [Ref. (4)].

Valores de los criterios de personal	Criticidad Alta	Criticidad Media	Criticidad Baja
(c1) Experiencia del usuario	1	0	0
(c2) Expresividad del usuario	1	0	0
(c3) Experiencia del desarrollador en el ámbito de la aplicación	0	1	0
(c4) Experiencia del desarrollador en el entorno SW	0	0	1
Valores de los criterios del problema			
(c5) Madurez de la aplicación	0	1	0
(c6) Complejidad del problema	0	1	0
(c7) Requerimientos de funcionalidad parcial	1	0	0
(c8) Frecuencia de cambios	0	1	0
(c9) Magnitud de los cambios	0	1	0
Valores de los criterios del producto			
(c10) Tamaño del producto	0	1	0
(c11) Complejidad del producto	1	0	0

(c12) Requerimientos cualitativos	0	1	0
(c13) Requerimientos de interfaz de usuario	1	0	0
Valores de los criterios del recurso			
(c14) Financiación	-	-	-
(c15) Disponibilidad de fondos	-	-	-
(c16) Perfil del personal	0	1	0
(c17) Disponibilidad de personal	-	-	-
(c18) Accesibilidad a los usuarios	1	0	0
Valores de los criterios organizacionales			
(c19) Gestión de compatibilidad	0	0	1
(c20) Garantía de calidad y metodología administrativa	-	-	-

TABLA 2. *CRITERIOS DEL PROYECTO.*

Modelo de Proceso	Clasificación	
NIVEL 3		
Convencional	5	[Anexo A.]
Incremental	**13**	[Anexo B.]
Evolutivo	12	[Anexo C.]
NIVEL 2		
Cascada	6	[Anexo D.]
Híbrido	**10**	[Anexo E.]
Especificación Operacional	8	[Anexo F.]

TABLA 3. *CLASIFICACIÓN.*

Traducido del inglés [Ref. (5)], *"El desarrollo software basado en incrementos* (**Figura 18**) *ofrece productos por versiones donde cada versión provee funcionalidad extra o modificada en comparación con la versión anterior. Las entregas incrementales de productos software proveen valor de negocio más pronto y permiten una rápida recepción de feedback por parte del cliente. Cada versión es una colección de características que conforman un sistema completo que será de valor para el cliente en el momento de la entrega. Este valor puede cambiar a lo largo del tiempo."*

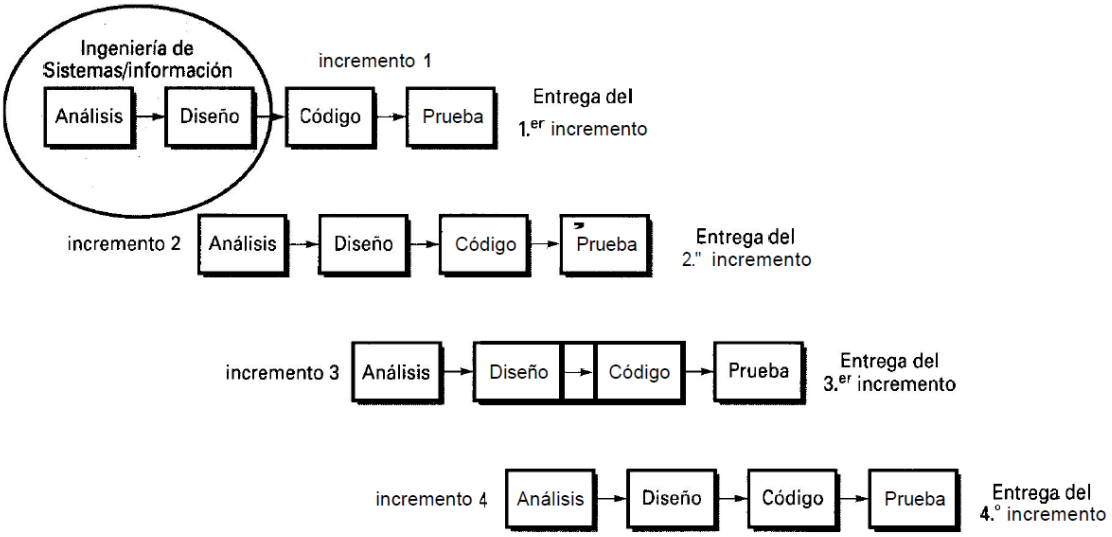

FIGURA 18. *MODELO INCREMENTAL.*

Con este tipo de modelos conseguimos que, aunque se avance más despacio que con un modelo convencional en cascada, el trato con el cliente sea más cercano y continuado, permitiendo de este modo que el cliente opine y determine si el proyecto va por buen camino y cumple con sus expectativas, o si por el contrario, el trabajo realizado no se corresponde con lo que desea recibir del sistema.

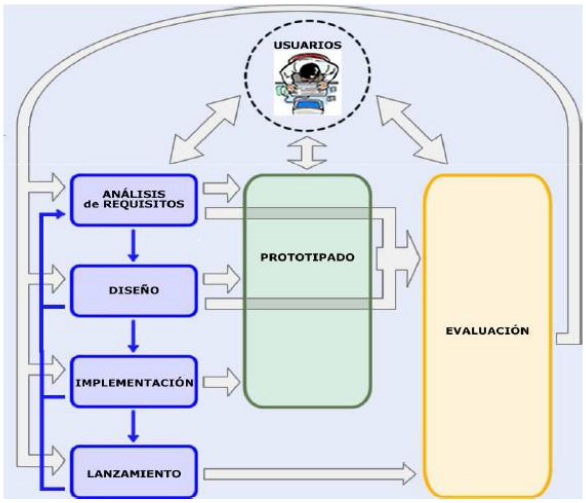

FIGURA 19. *MODELO HÍBRIDO DE PROTOTIPOS.*

De este modo conseguiremos que el cliente apruebe cada sección individualmente, manteniendo un progreso firme en el proyecto, y evitando un posible rechazo del cliente a una única entrega final. Es por esto, como anunciábamos en el apartado 1.2, que las pruebas realizadas se llevarán a cabo a medida que se vaya desarrollando el proyecto, obteniendo la validación y aceptación por parte del cliente.

Además, salvando las distancias, podemos decir que la metodología empleada se asimila al desarrollo ATDD [4] (Acceptance Test-Driven Development), que es una metodología de desarrollo basada en la comunicación entre cliente, desarrollador y 'tester'.

[4] http://en.wikipedia.org/wiki/Acceptance_test-driven_development

3. DISEÑO

Además de los requisitos educidos durante la fase de análisis, a la hora de realizar el diseño, se tuvo en cuenta el futuro mantenimiento de la aplicación. Por ello, se va realizar un pequeño análisis del concepto de mantenimiento.

Como determina el estándar ISO/IEC 14764 [Ref. (6)], el concepto de mantenimiento debe abordar:

- El alcance del mantenimiento del software.
- La definición del proceso de mantenimiento en general.
- La designación de los responsables de las tareas a realizar (en este caso, el autor del TFG realizará todas las tareas).
- Estimación de los costes.

Los principios a respetar para obtener un mantenimiento adecuado para una aplicación son:

- Claridad.
- Modularidad.
- Extensibilidad y flexibilidad de diseño.
- Buena documentación interna y externa.
- Llevar a cabo el mantenimiento de forma planificada.

Existen diversos problemas, que nos pueden surgir, relacionados con el mantenimiento, como la dificultad en el seguimiento de los cambios o la falta de previsión de futuros cambios durante la etapa de diseño.

Una de las maneras de evitar este tipo de problemas consiste en mejorar las labores de mantenimiento durante el desarrollo de software, mediante:

- Buena documentación.
- Uso de estándares.
- Uso de metodologías.
- Identificar posibles mejoras del producto y preverlas durante el desarrollo.

Teniendo en cuenta los requisitos y los conceptos que acabamos de explicar sobre el mantenimiento, se decidió agrupar los componentes software de la aplicación en tres grupos, empleando una arquitectura basada en el patrón MVC (Modelo-Vista-Controlador) [Ref. (7)] [Ref. (8)] [Ref. (9)]. Esta metodología nos permite cubrir la totalidad de conceptos y principios a tener en cuenta para obtener un mantenimiento adecuado.

Podemos definir cada componente del patrón MVC de la siguiente manera[5]:

- **Modelo**: *"es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'"*.

- **Controlador**: *"responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta de 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'"*.

- **Vista**: *"presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida"*.

[5] http://es.wikipedia.org/wiki/Modelo-vista-controlador

3.1. ARQUITECTURA Y TECNOLOGÍAS EMPLEADAS

En este apartado, se va a explicar la arquitectura software diseñada para el desarrollo de la aplicación, dando, en primer lugar, una descripción general del diseño arquitectónico, y posteriormente, explicando en detalle cada uno de los módulos que componen tal diseño, así como las tecnologías utilizadas para implementarlos.

3.1.1. DESCRIPCIÓN GENERAL DE LA ARQUITECTURA SOFTWARE

Como se puede observar en el diagrama de la **Figura 20**, en la arquitectura software diseñada para el desarrollo de la aplicación, existe una pequeña modificación respecto al estándar MVC. Se ha decidido, de cara a un mejor mantenimiento y reusabilidad en el futuro, dividir el controlador en dos partes claramente diferenciables que interactúan entre sí.

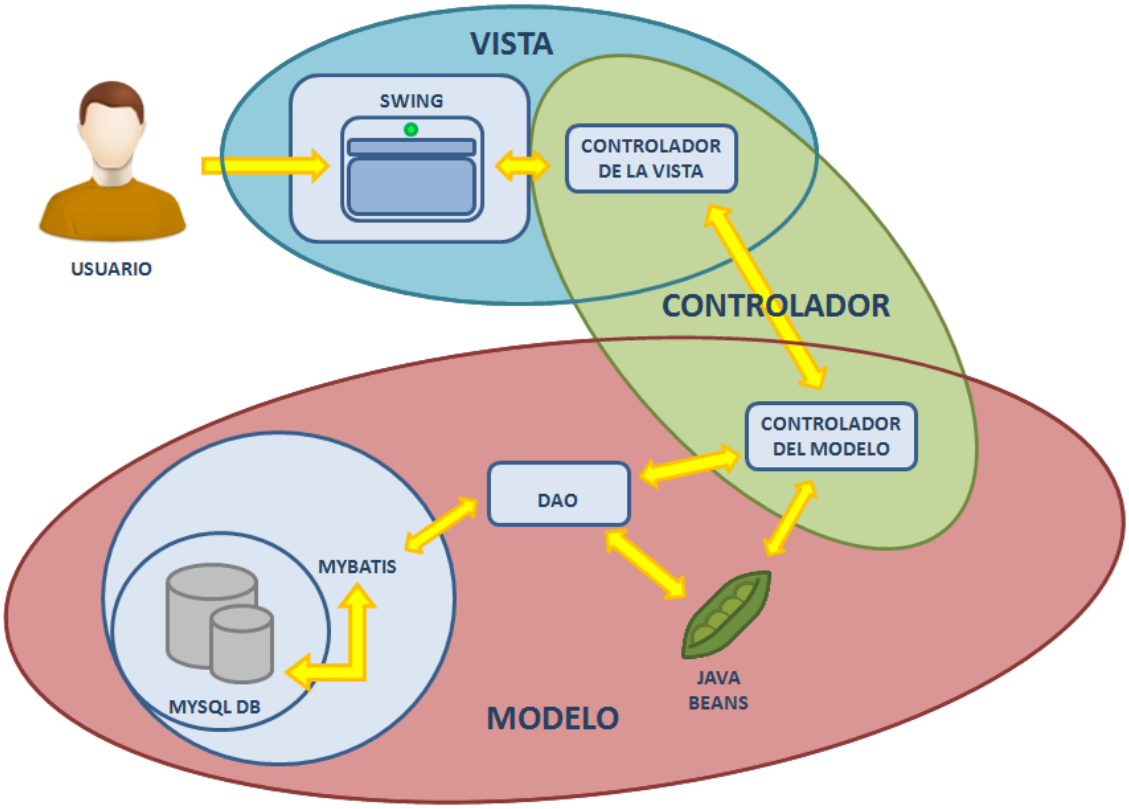

FIGURA 20. *DIAGRAMA DE LA ARQUITECTURA LÓGICA DEL SISTEMA.*

Con la división del controlador conseguimos que los cambios que se realicen en la vista afecten sólo a los "listener" y las funciones de actualización, de los elementos visuales, definidas en su controlador, manteniendo la funcionalidad del controlador del modelo intacta (ocurriría lo mismo en el sentido contrario). De este modo, se dota de una mayor independencia entre los elementos del sistema, exaltando tal característica del patrón MVC.

3.1.2. MODELO

El diseño se ha realizado desde dentro hacia fuera, es decir, comenzando por el modelo relacional de datos y terminando en la interfaz. Esta decisión se basa en poder dejar perfectamente definido el modelo de datos en este primer incremento (abarcado y desarrollado para el TFG) y dejando abierta la posibilidad de realizar cambios en la lógica de negocio y en la interfaz en futuros incrementos del proyecto. Otra opción habría sido partir del modelo de objetos (mediante un diagrama de clases), pero en nuestro caso nos resultó más natural recabar todos los datos que se registran en la empresa y construir la aplicación sobre los mismos.

A continuación se explicarán cada uno de los componentes, tanto físicos como lógicos, que soportan el modelo de datos de la aplicación:

- Base de datos relacional (MySQL)
- Capa de persistencia (MyBATIS)
- Capa de acceso a datos de alto nivel (DAOs)
- Encapsulado de los datos en objetos Java (BEANs)

BASE DE DATOS RELACIONAL

La base de datos se basa en un diseño de 32 tablas que almacenan toda la información generada en el día a día de la empresa.

Para algunas de las entidades que componen el esquema relacional, se ha distinguido entre dos tipos de tabla, una de catálogo o configuración y otra de registro. La tabla catálogo recoge los datos constantes de la entidad y la de registro almacena cada uno de los estados que vayan teniendo dichas entidades a lo largo del tiempo.

Antes de pasar a ver el esquema relacional así como la información almacenada en la base de datos, cabe decir que se planteó la posibilidad de implementar parte de la lógica de negocio en la propia base de datos mediante el empleo de triggers y procedimientos almacenados. Finalmente, y a pesar de saber que esta implementación supone un rendimiento más eficiente del manejo de los datos[6][7], se ha decidido desarrollar esta funcionalidad en la lógica de la aplicación para facilitar así su mantenimiento, reusabilidad y portabilidad.

Además, dado el volumen de datos que recogen los procesos de la empresa en cuestión, la mejora en rendimiento sería casi imperceptible, y sin embargo, es un

[6] http://social.msdn.microsoft.com/Forums/es-ES/manejar-lgica-de-negocio-en-store-procedure
[7] http://es.wikipedia.org/wiki/Procedimiento_almacenado

requisito imprescindible facilitar las labores de mejora y expansión de la aplicación en incrementos futuros.

Consultando la **Figura 21** podemos ver en detalle la totalidad de los campos que componen cada una de las tablas, así como sus claves primarias y foráneas.

Cabe destacar que en algunas de las tablas maestro que son referenciadas por otras tablas, se ha incluido un campo binario para la realización de **borrados lógicos** de los datos. Esta decisión se ha tomado dada la necesidad de mantener una referencia a los datos que no pueden borrarse. Por ejemplo, si se borra un producto para que deje de aparecer en la aplicación, no se puede eliminar por completo si se hace referencia a él desde una venta, ya que si no, se perderían datos que deben mantenerse como parte de la venta.

Además, excepto en las tablas puramente relacionales, se ha decido incluir un **identificador numérico** único que componga por sí mismo una clave primaria. La finalidad de esta decisión de diseño reside en la mayor eficiencia, a la hora de trabajar con la tabla, que supone hacerlo si cuenta con una clave primaria numérica. Este identificador es transparente para el usuario final, ya que también se provee de un código alfanumérico sobre el cual sí tiene control el usuario del sistema.

En el caso de las tablas de registro o histórico, además del id del registro de la entidad maestro referenciado (un producto, un cliente, etc), se emplea un ***timestamp*** como parte de la clave primaria.

Antes de ver los diferentes grupos de tablas empleados, cabe decir que como gestor de base de datos se ha decido emplear MySQL y como cliente MySQL Workbench 6.1 por ser software libre y disponer de un amplio subconjunto del lenguaje SQL (Structured Query Language), que es del que disponemos de un mayor conocimiento.

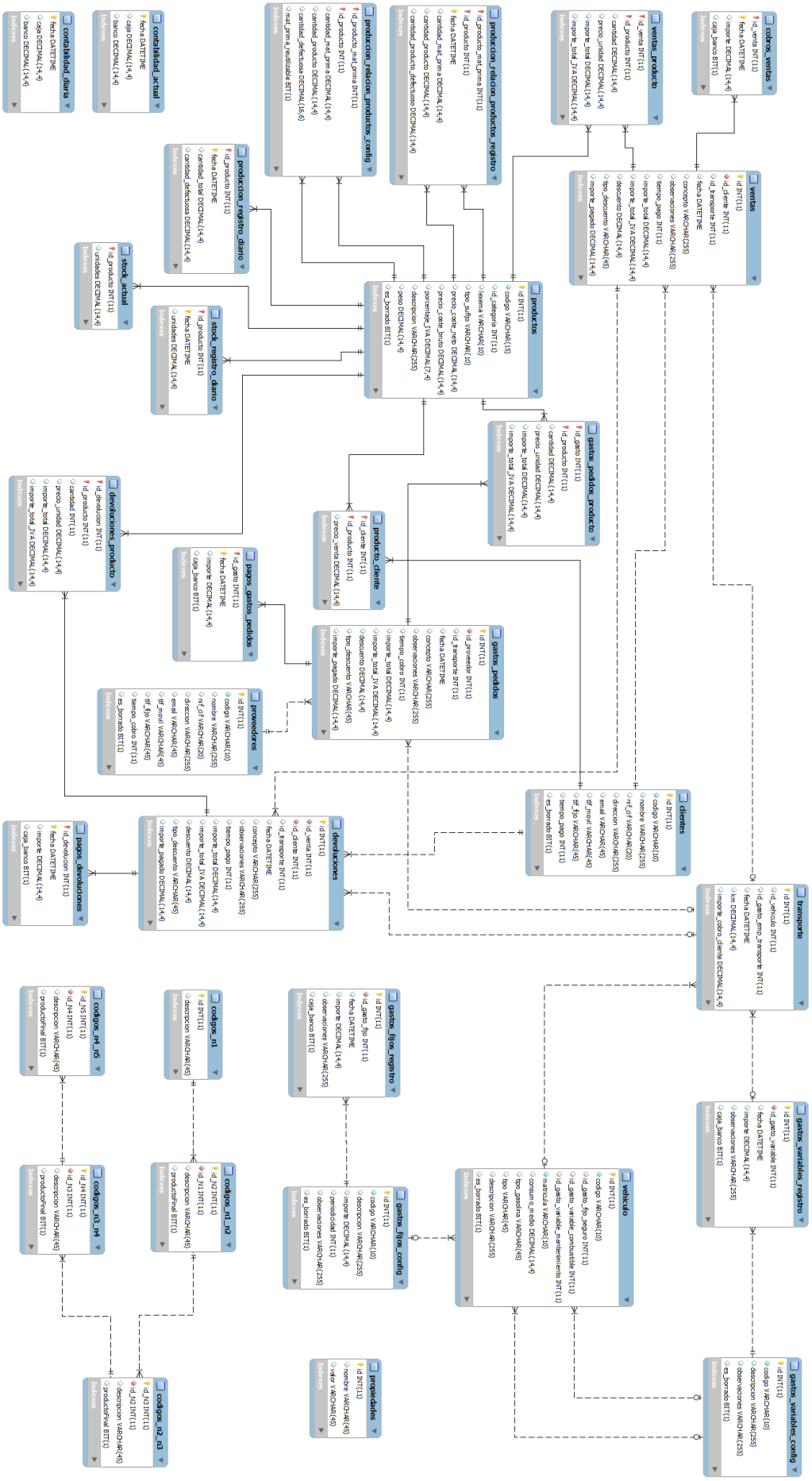

FIGURA 21. *ESQUEMA RELACIONAL.*

Dentro de la variedad de tablas empleadas en el modelo, podemos distinguir, para explicar de un modo más organizado y resumido su diseño, los siguientes grupos:

Productos, stock y producción.

En esta sección podemos distinguir 6 tablas diferentes:

- **Productos**
- **Stock_actual**
- **Stock_registro_diario**
- **Producción_relacion_productos_config**
- **Producción_relacion_productos_registro**
- **Producción_registro_diario**

Tales tablas nos permiten almacenar la ficha completa de un producto, el stock del mismo que se va manteniendo en almacén siguiendo una línea temporal, el histórico de producción que se va ejerciendo sobre tal producto y las relaciones de coste de producción del producto, en cuanto a cantidad de materias primas se refiere. Para el stock se hace una distinción del stock actual y el histórico en pos de favorecer los accesos a base de datos que se realizan con mayor frecuencia. Para organizar los productos se les asigna un código de la categoría o subcategoría a la que pertenecen.

Categorías y subcategorías de productos.

Se cuenta con 5 tablas de relación de categorías y subcategorías por niveles:

- **Códigos_N1**
- **Códigos_N1_N2**
- **Códigos_N2_N3**
- **Códigos_N3_N4**
- **Códigos_N4_N5**

Clientes y proveedores.

Para almacenar los datos relacionados con los clientes y proveedores se emplean las siguientes tablas:

- **Clientes**
- **Producto_cliente**
- **Proveedores**

Las tablas de *clientes* y *proveedores* son muy similares y se encargan de gestionar toda la información relativa a las fichas de las entidades que representan. La tabla *producto_cliente* permite albergar un catálogo de precios propio para cada cliente, determinando para un cliente concreto, el precio por defecto de un producto concreto.

Contabilidad, gastos fijos y gastos variables.

Tanto para contabilidad, como para gastos fijos y variables, se emplean 2 tablas:

- **Contabilidad_actual**
- **Contabilidad_diaria**
- **Gastos_fijos_config**
- **Gastos_fijos_registro**
- **Gastos_variables_config**
- **Gastos_variables_registro**

En el caso de la contabilidad, la finalidad de emplear 2 tablas es la misma que en el stock, tener una tabla para los accesos de mayor frecuencia y otra que almacena todo el registro.

En el caso de los gastos fijos y variables, la tabla de configuración contiene los datos propios de cada tipo de gasto contemplado por el sistema, así como la periodicidad de los mismos en el caso de los gastos fijos. La tabla de registro permite registrar cada uno de los gastos realizados en un día concreto. (Se debe tener en cuenta que los costes ocasionados por un pedido se consideran costes operativos y no gastos, y por ello se contabilizan en la tabla específica de pedidos).

Vehículos y transportes.

- **Vehículos**
- **Transportes**

La tabla de vehículos almacena todos los datos relacionados con un vehículo, así como una referencia a los gastos fijos y variables que puede ocasionar. En la de transportes se almacenan datos como los kilómetros realizados, la fecha y dependiendo de si el transporte se realiza con un vehículo propio de la empresa o mediante la contratación de un tercero, almacena el código del vehículo o el código del gasto.

Ventas, devoluciones y pedidos.

Contamos con 9 tablas para esta sección:

- **Ventas**
- **Venta_producto**
- **Cobros_venta**
- **Devoluciones**
- **Devolucion_producto**
- **Pagos_devolucion**
- **Pedidos**
- **Pedido_producto**
- **Pagos_pedido**

Para cada una se cuenta con 3 tablas diferentes, una para los datos generales de la venta, devolución o pedido, otra para los productos vendidos, devueltos o pedidos y una tercera para registrar los posibles pagos en los que se decida fragmentar el importe total.

Propiedades.

La tabla de **propiedades** se emplea para almacenar un valor para las preferencias o parámetros de configuración de la aplicación, como puede ser, seleccionar si se desea tener una contraseña de acceso, el valor de dicha contraseña, si se desea que se realicen las operaciones automáticas de actualización sobre la base de datos, elegir el estilo de la interfaz, etc.

Framework de persistencia vs SQL embebido.

A lo largo de la carrera se ha utilizado, generalmente, SQL embebido como medio de acceso a la base de datos desde la aplicación. Motivado por aprender una tecnología nueva, y recomendado por el tutor de este TFG, se decidió estudiar la posibilidad de utilizar un framework de persistencia, en concreto, MyBatis e Hibernate.

Tras analizar la documentación y teniendo en cuenta las características y opiniones en la web de ambos frameworks, pudimos darnos cuenta que la principal ventaja que un framework de persistencia nos ofrece, reside en la organización, claridad y limpieza con la que gestionamos el acceso a datos. Por ello, se decidió dar el paso a analizar qué framework de persistencia nos convenía más.

MyBatis vs Hibernate.

Para determinar qué framework nos convenía más utilizar, se comparó MyBatis con su principal competidor, Hibernate[8]. Para ver una comparativa más extensa entre diferentes tecnologías para el desarrollo de la capa de persistencia se puede consultar la referencia [Ref. (10)]. La **Tabla 4** muestra la comparativa entre ambos frameworks según la referencia [Ref. (11)].

Características	MyBatis	Hibernate
Simplicidad	La mejor	Buena
Solución ORM completa	Media	La mejor
Dependencia de SQL	Buena	Media
Soporte de lenguajes Query	Buena	Media
Rendimiento	La mejor	La mejor
Portabilidad entre diferentes bases de datos relacionales	Media	La mejor
Soporte de la comunidad y documentación	Media	Buena
Portabilidad entre aplicaciones no-Java	La mejor	Buena

TABLA 4. *COMPARATIVA MYBATIS VS HIBERNATE.*

Como podemos observar, ambas tecnologías obtienen una valoración muy similar, por lo que dependiendo de las necesidades del proyecto, será mejor una opción que la otra.

[8] http://hibernate.org/

En nuestro caso, las valoraciones de las características según los requisitos consensuados con el cliente, son:

- **Simplicidad:** dada la limitación de tiempo que un TFG requiere (300 horas), resulta primordial poder aprender a utilizar el framework en el menor tiempo posible, y en eso MyBatis es un claro vencedor por su sencillez. La única restricción reside en tener conocimientos SQL, los cuales se han adquirido a lo largo de la carrera.

- **Solución ORM (Object Relational Mapping) completa:** Hibernate mapea las tablas de la base de datos directamente a objetos, mientras que Ibatis mapea los resultados de las queries a objetos. Como se indicó en el apartado 3.1.1, el diseño de la arquitectura está basado en el modelo de datos y el dominio de objetos empleado está orientado a la lógica de negocio. Sobre estas condiciones, y al contrario de lo indicado previamente en la **Tabla 4**, en nuestro caso, MyBatis es la opción más adecuada.

- **Dependencia de SQL:** con Hibernate no es necesario tener conocimientos de SQL, ya que se encarga de hacerlo por nosotros. Con MyBatis tenemos un mayor control sobre las consultas, y con los conocimientos adecuados sobre SQL se puede conseguir un mayor rendimiento.

- **Soporte de lenguajes Query:** MyBatis soporta SQL al completo mientras que Hibernate dispone de un lenguaje propio HQL (Hibernate Query Language).

- **Rendimiento:** en la versión inicial de MyBatis (ibatis) no se ofrecía la posibilidad de emplear cachés de datos, por lo que estaba en desventaja con Hibernate. Sin embargo, en la versión actual de MyBatis (MyBatis 3) sí está disponible. En nuestro caso el rendimiento no supone un punto crítico dado el volumen de los datos utilizados, por lo que se ha decidido no emplear caché y limitarnos a las mejoras de rendimiento que MyBatis gestiona automáticamente mediante el uso de prepared statements. Además, el uso de una caché puede ocasionar problemas de consistencia de los datos empleados entre la aplicación, la caché y la base de datos.

- **Portabilidad entre diferentes bases de datos relacionales:** debido al empleo de un lenguaje propio, las migraciones en Hibernate resultan muy sencillas. En el caso de MyBatis, esta portabilidad se ve condicionada por la portabilidad propia de las consultas SQL utilizadas. En nuestro caso, las consultas SQL

empleadas en la aplicación son consultas estándar, por lo que la portabilidad entre diferentes motores de bases de datos no supone una desventaja real.

- **Soporte de la comunidad y documentación:** Hibernate cuenta con una mayor comunidad que lo soporta, sin embargo, la sencillez de MyBatis permite que con la documentación oficial ofrecida se puedan solventar los problemas o dudas más comunes que puedan surgir durante su utilización.

- **Portabilidad a aplicaciones no-Java:** aunque MyBatis proporciona alguna posibilidad más, la principal alternativa que se tendría en cuenta para el futuro sería .NET y está cubierta por ambas tecnologías.

En vista de los resultados comparativos descritos entre ambos frameworks, se decidió utilizar MyBatis. Por último, aunque en nuestro caso no se ha empleado, el framework de MyBatis se encuentra disponible como plugin para Eclipse, y dispone de módulos de integración con Spring Framework y Google Guice.

MyBatis.

"MyBatis[9] *es un framework de persistencia que soporta SQL, procedimientos almacenados y mapeos avanzados. MyBatis elimina casi todo el código JDBC, el establecimiento manual de los parámetros y la obtención de resultados. Puede configurarse con XML o anotaciones y permite mapear mapas y POJOs (Plain Old Java Objects) con registros de base de datos".*

Como acabamos de leer, MyBatis puede utilizarse con ficheros XML (eXtensible Markup Language) y anotaciones. En nuestro caso, se han empleado ficheros XML como medio de configuración, y poniéndonos más estrictos con la definición del framework, se han empleado BEANs en lugar de POJOs (Más adelante, en esta misma sección, veremos cuál es la diferencia).

Los elementos necesarios para emplear MyBatis son:

- **Librería**: Incluir el archivo .jar de la versión que deseemos utilizar.
- **Fichero de propiedades:** contiene la cadena de conexión con la base de datos, es decir, los valores para el driver (jdbc), username, password y url.
- **Fichero de configuración xml**: además de hacer referencia a las propiedades previamente mencionadas, contiene la relación de los diferentes mappers

[9] http://mybatis.github.io/mybatis-3/es/

(fichero xml que permite mapear los BEAN con los parámetros de las consultas a ejecutar).

- **SqlSessionFactory**: esta clase, a través del fichero de configuración, permite establecer la conexión con la base de datos y ofrece un acceso a la misma.

Mappers.

Estos ficheros nos permiten, de una manera sencilla y estructurada, mapear los datos de nuestra base de datos con los objetos empleados en la lógica de negocio.

En nuestro caso, cada mapper utilizado se estructura de la siguiente manera:

- **Cabecera** del documento: en la **Figura 22** podemos ver un ejemplo del mapper de la tabla de clientes.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <!DOCTYPE mapper
3     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4       "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6   <mapper namespace="Clientes">
```

FIGURA 22. *CABECERA MAPPER CLIENTES.*

- **Namespace:** identifica el mapper para ser utilizado posteriormente.
- **ResultMap:** se emplea para realizar el mapeo entre las columnas de los resultados devueltos por la base de datos al ejecutar una consulta y los campos del objeto BEAN que se utilizará para devolver el resultado al DAO correspondiente. En la figura podemos ver un ejemplo.

```
6   <mapper namespace="Clientes">
7
8       <resultMap id="result" type="com.contaaridos.bean.ClienteBEAN">
9           <result property="id" column="ID"/>
10          <result property="codigo" column="CODIGO"/>
11          <result property="nombre" column="NOMBRE"/>
12          <result property="nifCif" column="NIF_CIF"/>
13          <result property="direccion" column="DIRECCION"/>
14          <result property="email" column="EMAIL"/>
15          <result property="tlfMovil" column="TLF_MOVIL"/>
16          <result property="tlfFijo" column="TLF_FIJO"/>
17          <result property="tiempoPago" column="TIEMPO_PAGO"/>
18          <result property="esBorrado" column="ES_BORRADO"/>
19      </resultMap>
```

FIGURA 23. *RESULTMAP DEL MAPPER CLIENTES.*

- **Consultas:** existen 4 etiquetas diferentes para clasificar las consultas que se pueden realizar contra la base de datos (select, insert, update y delete). Todas

las consultas deben tener un atributo id que las identifique. Además, dependiendo de la consulta, se puede tener un argumento de entrada indicando el tipo en el atributo parameterType. Dicho atributo puede ser un tipo básico propio de Java o un Objeto que hayamos creado en la aplicación, como en el caso de los BEANs. Si deseamos utilizar diferentes parámetros en una misma consulta, podemos emplear un HashMap.

```
96    <update id="setAsDeletedByCodigo" parameterType="String">
97        UPDATE CLIENTES
98        SET
99            ES_BORRADO = 1
100       WHERE CODIGO = #{codigo};
101   </update>
```

FIGURA 24. *EJEMPLO UTILIZACIÓN PARAMETERTYPE.*

Del mismo modo que los argumentos de entrada, los de salida se indican mediante un resultType o resultMap, siendo el primero el empleado para consultas que sólo devuelven un registro, y el segundo para devolver más de uno. Dentro de las consultas se permite la utilización de instrucciones de control para consultar el valor de los parámetros antes de emplearlos.

```
45    <select id="getByParameters" parameterType="HashMap" resultMap="result">
46        SELECT * FROM CLIENTES WHERE ES_BORRADO = 0
47        <if test="codigo != null">
48            AND CODIGO = #{codigo}
49        </if>
50        <if test="nombre != null">
51            AND NOMBRE = #{nombre}
52        </if>
53        <if test="nifCif != null">
54            AND NIF_CIF = #{nifCif}
55        </if>
56    </select>
```

FIGURA 25. *EJEMPLO INSTRUCCIONES DE CONTROL.*

Para una información más detallada de la configuración y el uso de los mapper, consultar la documentación oficial del framework [Ref. (12)] [Ref. (13)].

DATA ACCESS OBJECTS Y JAVA BEANS

Un Data Access Object (DAO) es un elemento que define una nueva capa de abstracción entre la de persistencia (en nuestro caso, el framework MyBatis) y las capas superiores de la aplicación, en el caso de nuestra arquitectura, el controlador del modelo. Para comprender los mecanismos de comunicación entre las distintas capas de nuestra aplicación, en el apartado 3.1.6 se muestra un diagrama de secuencia ilustrativo con las acciones que se deben llevar a cabo para insertar los datos de un nuevo cliente en la base de datos. Con el patrón DAO continuamos con la división del sistema en capas independientes entre sí, facilitando las posibilidades de realizar cambios en un futuro.

Junto al patrón DAO suele darse otro patrón conocido como Data Transfer Object (DTO). Como su propio nombre indica, un DTO es un objeto que sirve para transportar datos. Dentro de los DTO, podemos encontrarnos con los términos POJO (Plain Old Java Object) y BEAN. La principal diferencia reside en que un BEAN sigue una convención que determina que el objeto se debe componer de una serie de atributos privados, proveer métodos públicos que permitan consultarlos (getters) y modificarlos (setters) y proveer un constructor público por defecto. Por el contrario, un POJO no sigue ningún tipo de regla, pero la finalidad tanto de un POJO como de un BEAN consiste en proveer un encapsulamiento de datos en un objeto. El concepto de "elemento de transporte de datos" que hay detrás de un DTO se puede observar en la clase ClienteBEAN del diagrama de secuencia del apartado 3.1.6. Los objetos de esta clase sirven de mecanismo de comunicación entre los componentes de las distintas capas de nuestra arquitectura.

Con el uso de DAOs y BEANs conseguimos que la procedencia de los datos sea transparente para el controlador del modelo, ya que éste sólo entiende de BEANs y no conoce ni necesita saber si los datos provienen de un fichero de texto, una base de datos o de un servicio web.

La funcionalidad que ofrecen los DAO utilizados en el sistema se basa en el acrónimo CRUD (**C**reate, **R**ead, **U**pdate and **D**elete), que en términos más cercanos a sentencias SQL se traduce en el conjunto de operaciones que podemos realizar de consulta (select), inserción (insert), actualización (update) y borrado de datos (delete).

La principal desventaja del patrón DAO reside en el hecho de aumentar la cantidad de código que se tiene que desarrollar. Además, aumenta las líneas de código en tiempo de ejecución, por lo que podemos ver damnificado el rendimiento de la aplicación. Como ya se ha comentado en ocasiones anteriores, nuestra aplicación no es

crítica en temas de rendimiento, por lo que es preferible asumir ese coste de rendimiento en detrimento de un mejor mantenimiento.

En el incremento que el TFG abarca del proyecto completo, se ha decidido no utilizarlo dado que las necesidades actuales no lo requieren, pero una práctica típica, que podría ser un incremento futuro de la aplicación, es combinar el patrón DAO con el patrón Singleton. Esta combinación busca reducir el número de DAOs que se mantienen en memoria a uno por clase. De esta manera conseguimos que si hay múltiples usuarios utilizando el mismo DAO sólo se tenga una instancia del mismo en memoria.

Por último, decir que cada tabla en base de datos cuenta con su correspondiente BEAN y DAO en las capas superiores del modelo de datos. A continuación, la **Figura 26** muestra el DAO utilizado en la tabla de clientes para tener una idea de la funcionalidad que ofrecen y cómo se estructuran.

```
20  public class ClientesDAO {
21
22      private final SqlSessionFactory sqlSessionFactory;
23
24      public ClientesDAO() {...3 lines }
27
28      @SuppressWarnings({"unchecked", "ConvertToTryWithResources"})
29      public List<ClienteBEAN> selectAll() {...9 lines }
38
39      @SuppressWarnings({"unchecked", "ConvertToTryWithResources"})
40      private List<ClienteBEAN> selectAllPrivate() {...9 lines }
49
50      @SuppressWarnings("ConvertToTryWithResources")
51      public ClienteBEAN selectByCodigo(String codigo) {...9 lines }
60
61      @SuppressWarnings("ConvertToTryWithResources")
62      private ClienteBEAN selectByCodigoPrivate(String codigo) {...9 lines }
71
72      @SuppressWarnings("ConvertToTryWithResources")
73      public ClienteBEAN selectById(int id) {...9 lines }
82
83      @SuppressWarnings("ConvertToTryWithResources")
84      private ClienteBEAN selectByIdPrivate(int id) {...9 lines }
93
94      @SuppressWarnings({"unchecked", "ConvertToTryWithResources"})
95      public List<ClienteBEAN> selectByParameters(HashMap<String, Object> parameters) {...9 lines }
104
105     /* Este update mantiene las PK y cambia los demas datos */
106     public int update(ClienteBEAN cli) {...12 lines }
118
119     /* Este update mantiene las PK y cambia los demas datos */
120     private int updatePrivate(ClienteBEAN cli) {...12 lines }
132
133     public int insert(ClienteBEAN cli) {...16 lines }
149
150     public void deleteByCodigo(String codigo) {...9 lines }
159
160     public void deleteById(int id) {...9 lines }
169
170     private void deleteByCodigoPrivate(String codigo) {...9 lines }
179
180     private void deleteByIdPrivate(int id) {...9 lines }
189 }
```

FIGURA 26. *DAO DE CLIENTES.*

3.1.3. CONTROLADOR

CONTROLADOR DEL MODELO (LÓGICA DE NEGOCIO)

Como ya se anunció en el apartado 3.1.1, el controlador del modelo implementa la lógica de negocio de la aplicación. Se divide en varias clases o controladores encargados de traducir los diferentes tipos de acciones que se solicitan desde la vista a operaciones sobre los datos. Para ello, hacen uso de los DAOs definidos en el nivel inmediatamente inferior. De nuevo, en el diagrama de secuencia del apartado 3.1.6 podemos observar cómo tiene lugar este proceso.

Por ejemplo, la acción insertar nueva venta, se traduce en una inserción del nuevo registro creado en la tabla de ventas, y si finaliza de manera satisfactoria, se realiza una actualización de la contabilidad actual (que ha incrementado como resultado de realizar una venta y efectuar un cobro) y del stock de los productos vendidos (que ha disminuido). El controlador correspondiente se encarga de "controlar" esta secuencia de acciones y enlazarlas con la capa de presentación.

A continuación se van a explicar en detalle aquellas acciones que suponen una secuencia de operaciones sobre los datos mediante el uso de más de un DAO. Además de estas acciones, los controladores ofrecen a las capas superiores todas las operaciones básicas (CRUD) ofrecidas por los DAO, consiguiendo de esta forma un alto grado de abstracción. Dichos métodos se omiten en los apartados siguientes para no sobrecargar la información explicada en el documento.

StockCONTROLLER y ContabilidadCONTROLLER.

Las operaciones básicas sobre el stock las recoge el propio DAO, pero en este módulo se ofrece una interfaz de operaciones que tienen en cuenta la continuidad temporal.

- **Insertar**: realiza una inserción en el registro y posteriormente comprueba si hay registros con fecha posterior. En ese caso, aumenta la cantidad de unidades de los mismos con la cantidad indicada en el registro a insertar.

- **Borrar**: mismo concepto que para insertar pero a la inversa; si hay registros posteriores se resta la cantidad del registro a borrar.

- **Actualizar**: borrar el registro modificado e insertar con los nuevos datos.

En el caso de la contabilidad el proceso es el mismo que en el stock, pero en lugar de hablar de cantidades de producto, se trabaja con dinero en caja o en el banco.

ProduccionRegistroDiarioCONTROLLER.

Tres métodos:

- **Insertar**: recibe un BEAN con los datos del registro a insertar. En primer lugar, actualiza el stock actual del producto en cuestión añadiendo a la cantidad de unidades existente el total de unidades producidas. Además, consultando la relación cantidad de materia prima – cantidad de producto elaborado, reduce del stock el número de unidades empleadas para producir, y en caso de ser una materia prima reutilizable, vuelve a sumar la cantidad empleada para producir las unidades defectuosas de la partida de producción.

- **Borrar**: recibe un id de producto y una fecha para identificar el registro a borrar. Sigue el proceso inverso de insertar, restando al stock las unidades producidas y sumando la cantidad de materia prima.

- **Actualizar**: recibe un BEAN con los nuevos datos. En primer lugar borra y posteriormente inserta los datos actualizados.

ProduccionRelacionProductosRegistroCONTROLLER.

La finalidad de este módulo reside en actualizar la tabla de configuración cuando se producen modificaciones en el registro:

- **Insertar**: inserta el nuevo registro, calcula las medias, teniendo ya en cuenta los nuevos valores, de los registros asociados a la misma relación materia prima – producto elaborado, y realiza una actualización de los valores medios de la tabla de configuración.

- **Borrar**: borra el registro indicado y realiza el cálculo de las medias para actualizar en la configuración.

- **Actualizar**: actualiza el registro indicado y realiza el cálculo de las medias para actualizar en la configuración.

GastosFijosRegistroCONTROLLER y GastosVariablesRegistroCONTROLLER .

La finalidad de este módulo se basa en mantener la consistencia de los datos de manera que las modificaciones se apliquen automáticamente en la contabilidad, siguiendo de nuevo una línea temporal:

- **Insertar**: inserta el nuevo registro y actualiza la contabilidad (en banco o caja según lo indique el registro insertado) posterior a la fecha del gasto.

- **Borrar**: borra el registro indicado y realiza la actualización correspondiente de la contabilidad.

- **Actualizar**: actualiza el registro indicado y realiza la actualización correspondiente de la contabilidad.

VehiculoCONTROLLER.

- **Borrar**: comprueba si hay un gasto fijo por el seguro del vehículo y hace un borrado lógico del mismo en la tabla de configuración. Lo mismo se hace para los gastos variables por combustible y mantenimiento.

VentasCONTROLLER.

El módulo de ventas, además de mantener la consistencia con los datos de las tablas de stock y contabilidad, recoge todas las operaciones que se realizan sobre la tabla de ventas, ventas_producto y cobro_ventas.

- **Insertar**: recibe todos los datos de la venta encapsulados en un BEAN, como los datos del cliente, el tiempo de pago, observaciones, concepto, fecha etc. Junto al BEAN se recibe una lista de los BEANs ventaProducto que conforman la venta y un BEAN cobroVenta con los datos del pago, en caso de producirse, parcial o completo que realice el cliente en el momento de la venta. Se inserta en primer lugar el registro de la tabla de ventas para obtener el id autogenerado para la misma. Con ese id completamos los BEANs ventaProducto recibidos, así como el cobroPago, y procedemos a insertarlos en sus correspondientes tablas por medio de los DAOs. Para finalizar, se realiza la actualización pertinente del stock de los productos vendidos así como el incremento generado en la contabilidad si el cobroVenta tiene un importe mayor que 0.

- **Borrar**: en primer lugar se realiza el borrado de los registros ventaProducto que componen la venta, con las pertinentes actualizaciones sobre el stock de dichos productos. El segundo paso es borrar los diferentes cobroVenta que haya asociados asociados a la venta, con sus respectivas actualizaciones en la contabilidad. El tercer paso es borrar la venta guardándonos el id del transporte asociado si tiene alguno, y el último paso, es borrar el transporte con las operaciones internas que deba realizar y que se verán más adelante.

- **Registrar nuevo / actualizar / borrar cobroVenta**: inserta / actualiza / borra el registro, actualiza la contabilidad y actualiza el importe pagado de la venta.

- **Registrar nuevo / actualizar / borrar ventaProducto**: inserta / actualiza / borra el registro en su tabla, actualiza el stock y actualiza el importe total de la venta.

- **Actualizar datos venta**: actualiza el registro pertinente de la tabla de ventas.

PedidosCONTROLLER.

Por no ser redundante, del módulo de pedidos sólo se va a explicar que funciona exactamente igual que el de ventas, con la salvedad que si en una venta se suma en contabilidad y se resta de stock, en los pedidos se resta en contabilidad y se suma en stock.

DevolucionCONTROLLER.

En el caso de las devoluciones, al igual que hemos hecho con los pedidos, diremos que el proceso es exactamente el mismo que en estos últimos, y explicaremos sólo las diferencias.

La primera diferencia es que una devolución siempre va asociada a una venta, y lo primero que se debe recibir es una referencia (el id) a la venta en cuestión. La otra diferencia es que se debe tener en cuenta que no se puede registrar una devolución mayor que la venta asociada. Por ejemplo, si hemos vendido 300 kg de arena, no se puede crear un devolucionProducto de 400 kg de arena, y en ese caso el sistema informará del error cometido.

TransporteCONTROLLER.

Se diferencia entre dos transportes posibles, uno realizado por los vehículos propios de la empresa y otro por una empresa contratada.

- **Insertar**: además de los datos que se registran de un transporte, como la fecha, tarifa o los kilómetros realizados, los cuales se reciben encapsulados en un BEAN, se recibe un indicador que determine si se trata de un vehículo propio o de un gasto variable, y en ambos casos, el identificador que lo referencie. Una vez recibidos los datos, se realiza la inserción en la tabla de transportes y se actualiza la venta, el pedido o la devolución para que contenga la referencia al transporte realizado así como para que actualice el importe total.

- **Borrar**: en primer lugar se debe borrar de la venta, pedido o devolución asociada la referencia al transporte en cuestión, así como actualizar el importe total. A continuación, se procede a borrar el registro y en el caso de ser un gasto variable y no un transporte con los vehículos propios de la empresa, se borra el registro que se hubiese generado.

- **Actualizar**: se borra el transporte y se inserta de nuevo con los datos actualizados.

CONTROLADOR DE LA VISTA

La funcionalidad que encapsulan los controladores de la vista se resume en realizar las funciones de los listener, es decir, dar respuesta a las acciones que realiza el usuario de la aplicación sobre los componentes de la interfaz gráfica, recoger los datos de la vista, invocar a las acciones que los controladores de negocio ofrecen y modificar los elementos visuales de la vista para que muestren los datos devueltos. Del mismo modo que en componentes anteriores, para comprender mejor el flujo de las acciones realizadas podemos consultar el diagrama de secuencia del apartado 3.1.6.

Las acciones realizadas cobrarán un mayor sentido cuando se expliquen las diferentes ventanas diseñadas para la aplicación, pero se reducen a:

Operaciones de carga y búsqueda.

- Cargar las listas que se muestran en las listas desplegables.
- Recoger los valores seleccionados en los filtros para emplearlos al hacer búsquedas y consultas contra la base de datos.

- Cargar los conjuntos de registros a mostrar en la tabla de resultados de los formularios.
- Cargar las categorías y subcategorías, así como los productos finales, a mostrar en la interfaz TPV utilizada para las ventas y pedidos.

Operaciones para la inserción.

- Recoger y validar los valores seleccionados en la tabla de inserción así como en los desplegables utilizados para la selección de claves foráneas.
- Llamar al método de inserción del controlador de modelo correspondiente.

Operaciones de edición.

- Recoger y validar los valores editados en la tabla de resultados de los formularios.
- Llamar al método de actualización del controlador de modelo correspondiente.
- En el caso del borrado de registros, se obtiene el registro seleccionado y se llama al método de borrado que ofrezca su controlador de modelo.

3.1.4. VISTA

Al tratarse de una aplicación local en Java, y dado el poco tiempo disponible para la realización del primero de los incrementos, se ha decido emplear Swing para la realización de la interfaz por haberse estudiado durante la carrera. Junto con Swing, se han utilizado dos librerías: Swingx[10] y Synthetica[11].

Swingx permite la utilización de algunos componentes y funcionalidades visuales que Swing no provee, enriqueciendo la capa de presentación y facilitando su usabilidad. Por ejemplo, JXBusyLabel (**Figura 27**), nos permite tener un label con una animación típica de carga, y AutoCompleteDecorator sirve para dotar de funcionalidad de autocompletado a los JComboBox utilizados (**Figura 28**).

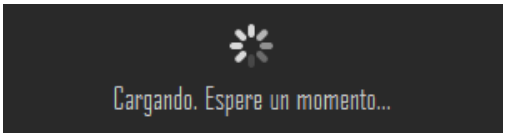

FIGURA 27. *EJEMPLO JXBUSYLABEL.*

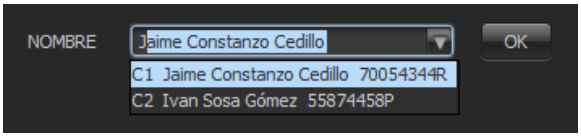

FIGURA 28. *EJEMPLO JCOMBOBOX.*

[10] https://swingx.java.net/
[11] http://www.jyloo.com/synthetica/

En el caso de Synthetica, se trata de una librería que además de ofrecer componentes visuales al igual que Swingx, como por ejemplo, DateCombobox (**Figura 29**), nos permite aplicar un tema de manera automática a los componentes básicos de Swing.

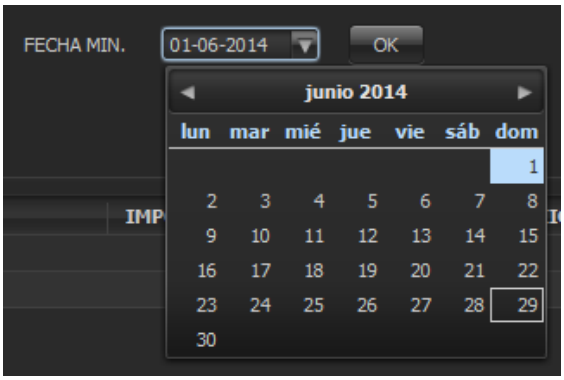

FIGURA 29. *EJEMPLO DATECOMBOBOX.*

Para utilizar diferentes temas, sólo se necesita cargar el archivo .jar de aquel que deseemos aplicar, y mediante una sencilla instrucción y el "import" correspondiente, podemos cambiar el aspecto de todos los componentes visuales de la aplicación de manera inmediata (**Figuras 30 y 31**).

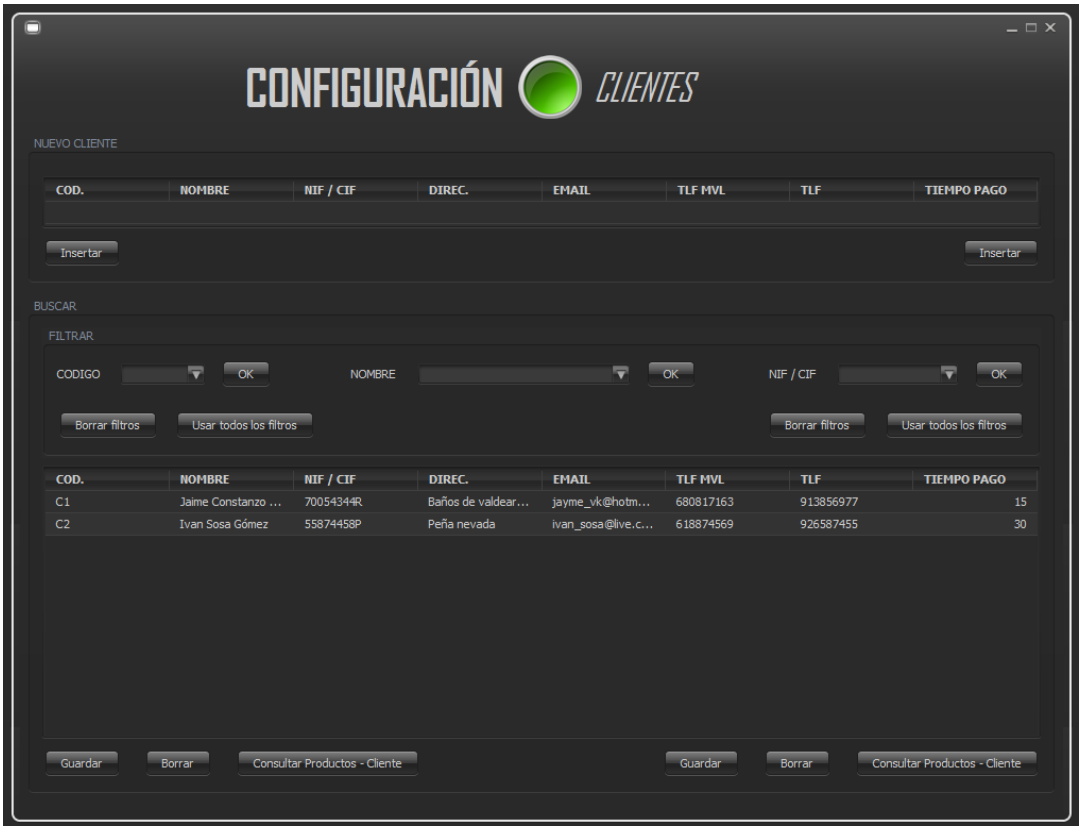

FIGURA 30. *FORMULARIO DE GESTIÓN DE CLIENTES I.*

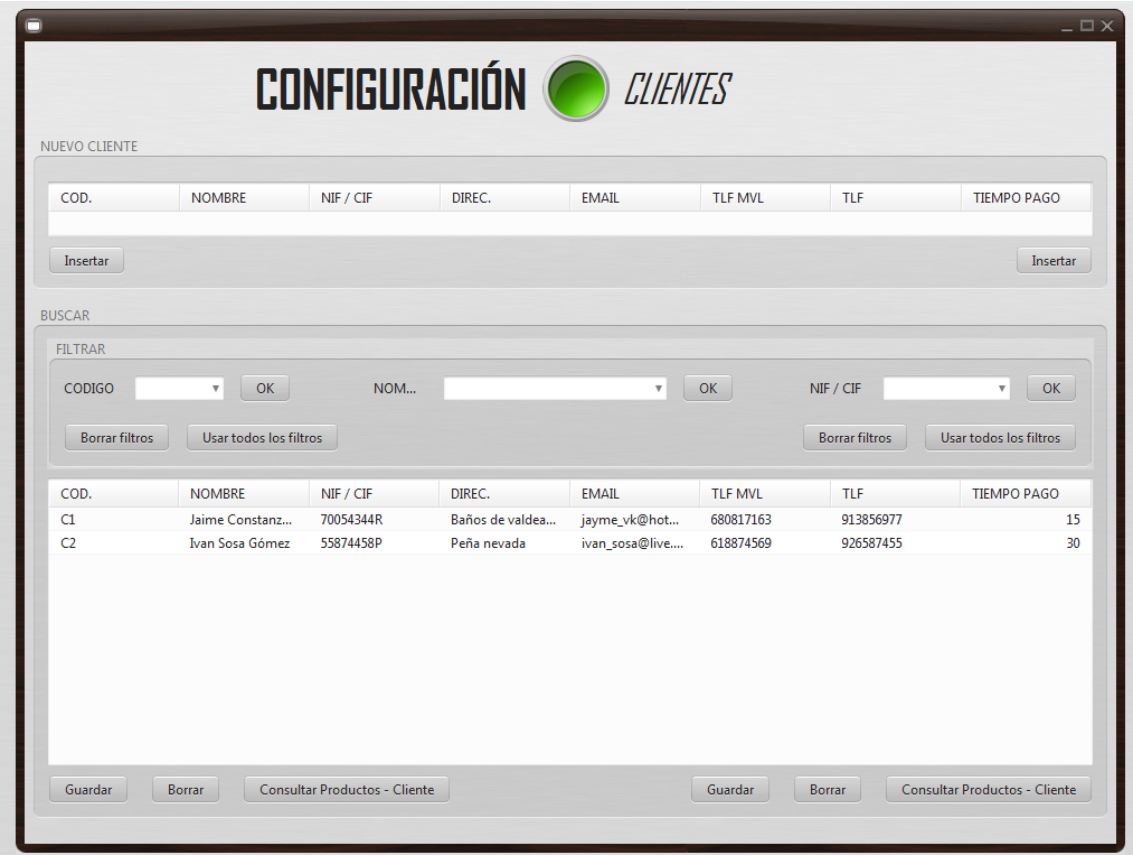

FIGURA 31. *FORMULARIO DE GESTIÓN DE CLIENTES II.*

Para la implementación de la interfaz se ha hecho uso del "drag and drop" que NetBeans nos ofrece. De este modo hemos podido implementar las diferentes ventanas diseñadas para la interfaz de usuario de una manera muy rápida y sencilla.

La variedad de ventanas diseñadas para la herramienta se puede clasificar en 3 grupos: menús de navegación, vistas de formulario y vistas con aspecto de TPV.

3.1.5. MENÚS DE NAVEGACIÓN

Atendiendo a los requisitos consensuados con el cliente, la interfaz debe ser fácil de utilizar, intuitiva y sencilla.

Uno de los aspectos a destacar en el diseño de la aplicación reside en la utilización de un botón (**Figura 32**) que permite volver al menú principal de la sección en la que estemos trabajando. Con esto conseguimos, sin necesidad de emplear barras de menú y resultando en una interfaz más limpia, poder llegar al menú principal de la aplicación desde cualquier ventana con un máximo de 3 pulsaciones.

FIGURA 32. *TÍTULO Y BOTÓN DE ATRÁS.*

Además, como se verá más en detalle en los apartados ***FORMULARIOS*** y ***TPV***, se ha provisto a la aplicación de botones de navegación rápida que permiten desplazarnos por las diferentes ventanas sin necesidad de pasar por los menús de navegación.

FORMULARIOS

Las ventanas empleadas para la gestión de datos y la configuración de la aplicación tienen un diseño a modo de formulario.

Dado que la funcionalidad de los formularios se basa en la toma de datos de manera clara y sencilla, se han empleado componentes visuales que reduzcan la posibilidad de cometer fallos por parte del usuario, como son el caso de los JComboBox (**Figura 28**) y los DateComboBox (**Figura 29**).

Los formularios de gestión de datos se caracterizan, en su mayoría, por tener en su parte superior una tabla de recogida de los datos a insertar en el nuevo registro, debajo un menú con los diferentes filtros para el motor de búsqueda y a continuación la tabla de resultados. En algunos casos, a este diseño se la añade un menú inferior de navegación rápida que permite navegar de manera inmediata a otro formulario que tenga relación con el activo. Podemos consultar las **Figuras 30 y 31** o el manual de usuario (**Anexo G**) para un mayor detalle.

TPV

El movimiento rápido de registro de una nueva venta, así como de un pedido, consta de una interfaz similar a la empleada típicamente en los TPV. Como hemos mencionado, el término TPV lo aplicamos solamente a la interfaz desarrollada, ya que dicha ventana no se sincroniza con una pasarela de pago en este primer incremento.

Actualmente, los productos se representan mediante botones vacíos, pero en un futuro, dichos botones estarán ilustrados con una imagen del propio producto. Además, los productos se muestran organizados por categorías y, de existir, por subcategorías a diferentes niveles.

Tanto los paneles de las categorías como los que engloban al botón y el nombre del producto, se han diseñado de tal manera que la inclusión de un nuevo producto en base de datos se gestione automáticamente a la hora de representarlo en la vista.

3.1.6. DIAGRAMA DE SECUENCIA.

Para clarificar la secuencia de acciones que internamente realiza el sistema y que van desde la vista hasta el modelo y viceversa, pasando por cada una de las capas de la aplicación, se muestra en las **Figuras 33 y 34** el diagrama de secuencia relativo a la **inserción de un nuevo cliente**, el cual se empleará como ejemplo.

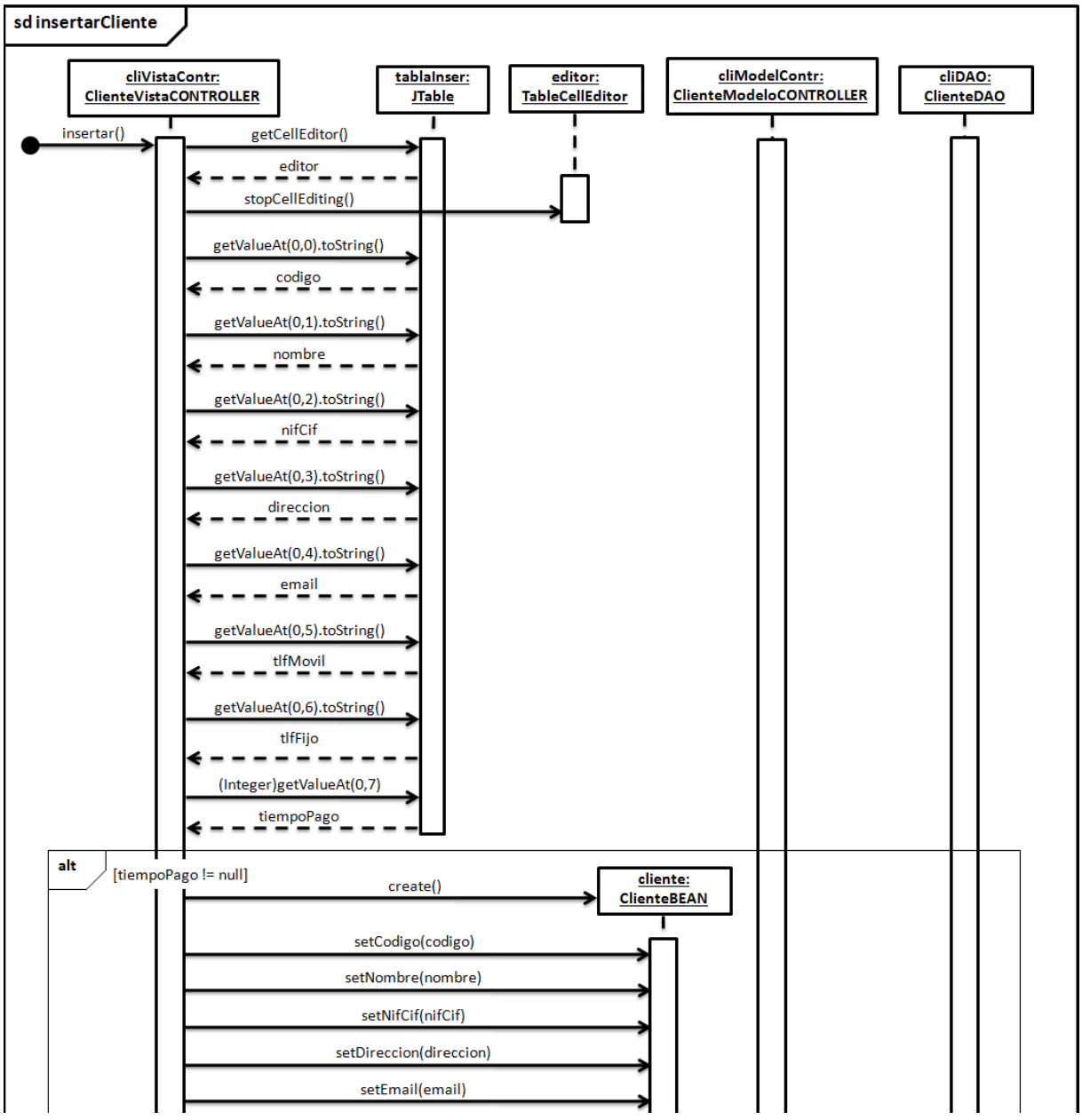

FIGURA 33. *DIAGRAMA DE SECUENCIA (PARTE I).*

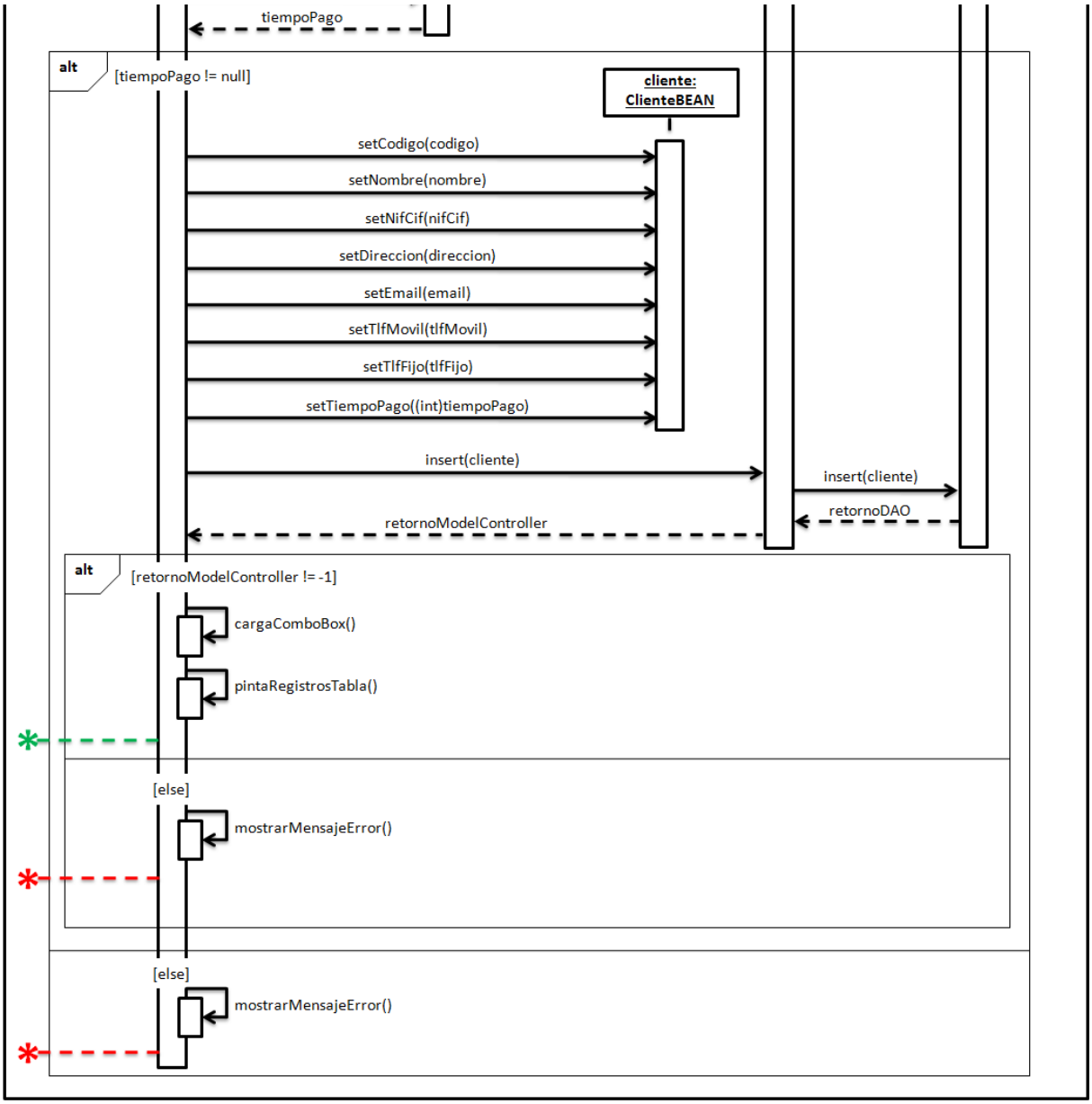

FIGURA 34. *DIAGRAMA DE SECUENCIA (PARTE II).*

ClienteDAO encapsula los mecanismos de acceso de la base de datos, haciendo que éstos sean transparentes para el controlador del modelo. De esta forma, en caso de cambiar el gestor de base de datos o la capa de persistencia, las modificaciones serían transparentes para el objeto cliente.

Los objetos de la clase ClienteBEAN, en este caso el objeto cliente, son objetos del tipo DTO, que sirven de elemento de comunicación entre las distintas capas de la aplicación. En el caso del ejemplo, el objeto se crea en el controlador de la vista para indicar al controlador del modelo los datos del cliente que se quiere crear. Este, a su vez, enviará este objeto al DAO correspondiente que será el que desencapsule la información e intente insertarla en la base de datos.

4. PRUEBAS

Lo primero que se debe destacar es que al haber utilizado un modelo de ciclo de vida en el que el usuario final tiene una participación, durante el diseño y desarrollo, muy activa, el plan de pruebas diseñado se ha realizado a medida que se iban desarrollando las diferentes funcionalidades implementadas.

4.1. ALCANCE DE LAS PRUEBAS

Se han realizado diferentes tipos de pruebas que validan y verifican el proyecto atendiendo a la funcionalidad, compatibilidad, contenido, usabilidad y accesibilidad.

4.1.1. FUNCIONALIDAD

La funcionalidad se prueba para descubrir errores que indiquen que no hay concordancia con los requisitos funcionales educidos. Junto con las pruebas de usabilidad, han sido las pruebas en las que más tiempo se ha dedicado.

4.1.2. COMPATIBILIDAD

En este primer incremento del proyecto, la única plataforma para la cual se ha consensuado el funcionamiento de la aplicación se reduce a Windows 7, y el entorno de producción no presenta diferencias notables al de desarrollo, por lo que no se ha realizado ningún tipo de prueba específica para la compatibilidad.

4.1.3. CONTENIDO

En el contexto de este proyecto, las pruebas de contenido no precisan de una gran relevancia, y por lo tanto, no se ha seguido un plan específico. Simplemente, durante las pruebas de validación y aceptación, se han solventado los errores sintácticos o semánticos descubiertos.

4.1.4. USABILIDAD

Prueba la facilidad con que las personas pueden utilizar una aplicación. Una forma de evaluar la usabilidad consiste en comprobar las 10 heurísticas de Nielsen [Ref. (14)] que permiten determinar si la aplicación cumple un nivel de usabilidad correcto. En nuestro caso, además de emplear las heurísticas de Nielsen para realizar una valoración más técnica, se ha diseñado un pequeño formulario (**Anexo H**), con algunas de esas heurísticas, que el propio cliente ha contestado.

4.1.5. ACCESIBILIDAD

Se prueba para garantizar el grado en el que todas las personas pueden acceder y utilizar una aplicación. En el ámbito de este proyecto, sólo existe un usuario final y la herramienta se ha desarrollado a medida para él, por lo que, teniendo en cuenta que el cliente ha participado activamente durante el diseño y desarrollo de la herramienta, no se han realizado pruebas de accesibilidad.

4.2. PLAN DE PRUEBAS

4.2.1. ESTRATEGIA DE PRUEBAS UTILIZADA

En nuestro caso, dada la poca disponibilidad de tiempo para la realización del proyecto, se ha acotado el plan de pruebas realizado a pruebas de integración, de validación y de usabilidad.

Pruebas de integración.

Empleadas para tener en cuenta la agrupación de módulos y el flujo de información entre las interfaces.

En este grupo tenemos las pruebas enfocadas para un nivel superior, es decir, las acciones de los controladores de modelo y controladores de la vista (así como los DAOs que emplean por debajo).

Para ello se han diseñado diferentes main de prueba que verifican que el funcionamiento de un módulo de código es correcto.

Pruebas de validación.

Se comprueba la concordancia respecto a los requisitos (funcionales y no funcionales) que se han acordado en la fase de definición del proyecto.

Como hemos mencionado anteriormente en el documento, la metodología de desarrollo que se ha empleado se ha basado en la continua validación y aceptación de las funcionalidades que componen el sistema colaborando conjuntamente con el cliente, de una manera similar a la metodología ATDD previamente explicada.

En las reuniones mantenidas con el cliente, además de explicarle la funcionalidad implementada, se le ha permitido utilizar la aplicación para poder verificar por sí mismo su correcto funcionamiento (pruebas alfa).

Se han probado los diferentes casos de uso (2.1.1) diseñados en la aplicación mediante sus respectivos casos de prueba, y podemos consultar algunos ejemplos de los mismos en el manual de usuario (**Anexo G**).

Pruebas de usabilidad.

Como anunciábamos en el alcance de las pruebas (4.1.4), para evaluar la usabilidad de la herramienta, hemos empleado las heurísticas de Nielsen, así como una simplificación de las mismas para recoger el grado de satisfacción del cliente mediante un sencillo cuestionario (**Anexo H**).

Decir de las heurísticas que sólo se han valorado aquellos criterios que no fuesen específicos de aplicaciones web.

REQUISITOS	CARACTERÍSITCAS	PUNTUACIÓN
1. Visibilidad del estado del sistema	Se debe mantener informado al usuario en todo momento sobre el estado en el que se encuentra el sistema.	6/10
Barra de direcciones url	En la barra navegadora podemos observar, según los parámetros introducidos previamente, el estado actual del sistema	No aplica
Ventana que indica los parámetros seleccionados	La página contiene en el panel izquierdo una lista de los criterios de búsqueda seleccionados	No aplica
Barras de progreso	Al realizar compras en el carrito, tales barras nos indicarán en qué etapa del proceso de compra nos encontramos	6/10
2. Adecuación entre el sistema y el mundo real	Lenguaje cercano al usuario, facilitando el uso de la aplicación. Información con orden lógico y natural.	9.66/10
Página de inicio	El orden lógico mostrado en la página principal es coherente. Lenguaje sencillo y fácilmente inteligible,	9/10
Menús	Los menús son sencillos, con una estructura adecuada a las necesidades de la aplicación.	9/10

Carrito de compra	El carrito de la compra es fácil de utilizar, está bien diseñado y resulta intuitivo.	10/10
3. Control y libertad para el usuario	Se ha de proveer al usuario de funciones para deshacer y rehacer las acciones realizadas.	9.33/10
Anular un pedido realizado	Una vez realizado un pedido es posible que el cliente se arrepienta de la realización del mismo	10/10
Modificar datos de usuario	En cualquier momento se podrán modificar los datos de un usuario registrado	10/10
Devolver el control al usuario en caso de error	Al producirse un error, el sistema deja en manos del cliente la libertad de poder abordarlo	8/10
4. Consistencia y cumplimiento de estándares	El usuario debe seguir las normas y convenciones de la plataforma sobre la que está implementando el sistema	8/10
Utilización de hojas de estilo	El uso de hojas de estilo facilita mantener la consistencia del diseño gráfico del sitio	No aplica
Usar de manera consistente el fraseo, las imágenes y las fuentes	Con ello conseguimos ofrecer una imagen de consistencia.	8/10
Preferir estilos por defecto para botones, barras de scroll, etc.	Facilitan al usuario el uso de la aplicación.	8/10
5. Prevención de errores	Es preferible evitar la ocurrencia de errores que desarrollar buenos mensajes de error.	8.33/10
Indicar los campos obligatorios en formularios	Al realizar los formularios de registro o de envío de un pedido, los campos obligatorios se marcan mediante un asterisco	6/10
Solicitar confirmación al usuario	Antes de realizar una operación que pueda producir una situación irreversible, se solicita al usuario que confirme que está de acuerdo	9/10

Eliminar acciones que induzcan a error	Teclear una opción entre muchas posibles en lugar de mostrar un menú desplegable.	10/10
6. Minimizar la carga de la memoria del usuario	El usuario no tiene por qué recordar información que recibió anteriormente.	8.66/10
Mostrar la información a través de objetos	Cuando un cliente selecciona un producto, no es necesario que se acuerde de los datos del mismo, ya que se guarda en el carrito.	10/10
Mostrar la información a través de acciones	Cuando un cliente selecciona un producto, no es necesario que se acuerde de los datos del mismo, ya que se guarda en el carrito.	10/10
Instrucciones de uso del sistema al alcance del usuario	De manera que facilite al usuario el uso de la aplicación sin tener que memorizarlo	6/10
7. Flexibilidad y eficiencia de uso	Los aceleradores permiten aumentar la velocidad de interacción para el usuario experto tal que el sistema pueda atraer a usuarios principiantes y experimentados.	9/10
Personalizar acciones frecuentes	Los usuarios registrados evitan repetir la introducción de datos al realizar un pago	10/10
Combinación de teclas	Por ejemplo, podemos realizar una búsqueda en la página pulsando Ctrl+F	No aplica
Carga rápida	La carga de la página debe ser lo más rápida posible, independientemente del tipo de conexión.	8/10
8. Estética y diseño minimalista	No debe contener información que sea irrelevante o que rara vez sea de utilidad.	8/10
Reducir el número de imágenes al mínimo	Cada imagen implica una descarga desde el servidor.	No aplica
Evitar información irrelevante	Reduce la visibilidad relativa de la información que sí es relevante	8/10

Información importante en la parte superior	Esta región siempre es visible en el navegador	No aplica
9. Ayuda para que el usuario reconozca, diagnostique y se recupere de los errores	Los mensajes de error deben expresarse en un lenguaje claro, indicar exactamente el problema y ser constructivos	7.66/10
Mensajes de error claros y concisos	Dichos mensajes deben ser claros, indicando exactamente el problema	9/10
Mensajes de error que propongan soluciones	Los mensajes constructivos facilitan al usuario resolver los errores	6/10
Resaltar los errores cometidos en un formulario	Resulta más fácil para el usuario ver dónde se ha equivocado que revisar todos los campos	8/10
10. Ayuda y documentación	Es preferible que un sistema no necesite de documentación, aunque puede ser necesario.	No aplica
Documentación fácil de encontrar	La documentación debe ser accesible fácilmente por el usuario, por ejemplo, mediante un enlace en la página principal.	No aplica
Centrada en las tareas del usuario	Tal documentación estará centrada en detallar las diversas funcionalidades que la aplicación ofrece a sus usuarios	No aplica
No ser muy extensa	La documentación será un breve manual de usuario / ayuda, por lo que deberá ser más bien breve	No aplica

4.3. CRITERIO DE FINALIZACIÓN

En la práctica se deja de realizar pruebas cuando se lleva un tiempo sin detectar errores o cuando no hay más tiempo hasta el día de entrega del producto.

En nuestro caso, al tratarse de un cliente conocido y dada la metodología de trabajo empleada, las últimas pruebas realizadas en este primer incremento han sido las pruebas de usabilidad. A partir de ahí, las pruebas a realizar consistirán en pruebas beta que realizará el propio cliente en su entorno de trabajo. Los errores que se detecten en dichas pruebas se solventarán como parte del mantenimiento correctivo.

5. PUESTA EN PRODUCCIÓN DEL PRIMER INCREMENTO

La puesta en producción se prevé llevarla a cabo una vez que se haya entregado y expuesto el presente TFG.

5.1. INSTALACIÓN

Para realizar la instalación, los requisitos necesarios son:

- Instalar Java SE 7 (o superior)
- Instalar MySQL Workbench 6.1 CE
- Cargar el dump de estructura de la base de datos
- Arrancar ContaAridos2014.jar

MySQL Workbench no sería imprescindible porque no es necesario tener un cliente de base de datos, sin embargo, facilitará las tareas de mantenimiento en el futuro.

5.2. FORMACIÓN Y MANUAL DE USUARIO

Dado que la metodología empleada se ha basado en una estrecha colaboración con el cliente para que fuese validando la herramienta a medida que se iba desarrollando, no ha sido necesario realizar un proceso de formación a posteriori.

El manual de usuario lo podemos consultar en el **Anexo G**.

6. INCREMENTOS FUTUROS Y MANTENIMIENTO

6.1. INCREMENTOS FUTUROS

Tras el análisis y diseño realizado hasta ahora, podemos determinar que como mínimo, habrá dos incrementos más a parte del implementado para este trabajo.

En el primero de ellos se incluirían las siguientes funcionalidades:

- **Informes**: informe de rendimientos de productos y clientes e informes de los estados financieros de la empresa.
- **Alertas**: sistema de notificaciones y alertas por exceso en los plazos de cobro y pago, avisos por los próximos pagos a realizar, etc.
- **Impresión de facturas**: permitir la impresión de facturas y presupuestos cuando se registra una venta o un pedido.
- **Sincronización con sistema de pago**: poder realizar pagos mediante una pasarela de pago sincronizada directamente con la interfaz TPV implementada.
- **Transportes con Google Maps**: cálculo de rutas para la realización de transportes, así como el seguimiento de los kilómetros realizados por los vehículos de la empresa de forma automática.

El tiempo previsto para la realización del incremento se estima en aproximadamente 400 horas, o lo que es lo mismo, 50 jornadas.

El segundo incremento englobaría todas las operaciones de análisis y rediseño necesarias para migrar el sistema a una aplicación web con servidor propio. El diseño de la arquitectura software que soporta la versión actual de la aplicación se realizó teniendo en cuenta esta posible migración. Gracias a la arquitectura MVC empleada en diferentes capas, la migración se podría realizar de una manera relativamente sencilla, siendo la vista el componente sobre el que más cambios tendrían que realizarse, y dejando el modelo, y su controlador, prácticamente intactos.

Para este último incremento no se ha realizado un análisis y diseño detallado de los cambios que se deberían efectuar, por lo que sólo es posible dar una estimación aproximada del esfuerzo necesario para llevarlo a cabo: 450 horas.

6.2. MANTENIMIENTO

Existen diversos tipos de mantenimiento, los cuales se clasifican en dos subgrupos: mantenimientos de corrección y de mejoras.

- ***Corrección***: atender al reporte de errores por parte del cliente o solventar el comportamiento indebido del producto en situaciones anteriormente no detectadas. Distinguimos entre dos tipos de mantenimiento de esta clase:
  - ***Correctivo***.
  - ***Preventivo***.
- ***Mejoras***: tipos de mantenimiento encargados de gestionar las mejoras o actualizaciones del producto. En esta categoría podemos encontrar los siguientes tipos:
  - ***Adaptativo***.
  - ***Perfectivo***.

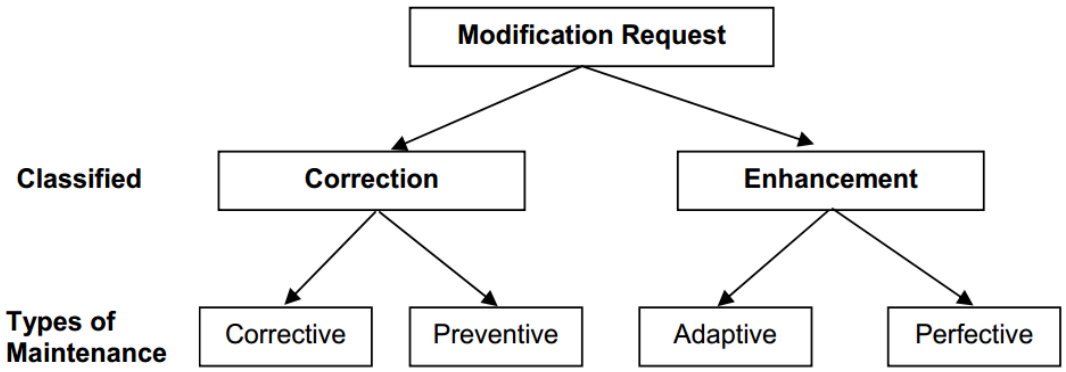

FIGURA 35. *ISO/IEC 14764:2006 (E). IEEE STD 14764-2006.*

6.2.1. MANTENIMIENTO CORRECTIVO

El mantenimiento correctivo [Ref. (15)] [Ref. (16)] de un proyecto software hace referencia a la reparación y depuración de posibles errores que puedan surgir en el producto después de su fase de desarrollo.

El objetivo de este mantenimiento es mantener el software funcional aún tras errores no previstos, impidiendo un acortamiento de la vida eficaz del mismo. Es la forma más básica de mantenimiento, que consiste en localizar errores o defectos y repararlos.

En nuestro caso, este tipo de mantenimiento será el que se realice a medida que se vaya utilizando la aplicación y el cliente (o el propio desarrollador) detecte errores que no se habían encontrado durante el plan de pruebas realizado. Tales correcciones se realizarían incorporándolas como parte del siguiente incremento.

6.2.2. MANTENIMIENTO PREVENTIVO

Supone modificar el sistema para mejorar sus propiedades (aumentando su ***calidad*** y/o ***mantenibilidad***) sin alterar sus especificaciones funcionales, manteniendo la ***eficiencia*** y ***fiabilidad*** del software.

En nuestro plan de mantenimiento no se contemplan acciones relacionadas con realizar un mantenimiento preventivo.

6.2.3. MANTENIMIENTO ADAPTATIVO

Mediante la realización de un mantenimiento adaptativo [Ref. (17)], se podrá modificar el sistema para acomodarlo a cambios físicos del entorno, como el traslado de la empresa a una sucursal diferente, la renovación de los equipos informáticos, los cambios en el sistema operativo, en el tipo de arquitectura, etc.

Dentro de las actividades que engloba el mantenimiento adaptativo se encuentran las actividades que serán necesarias para llevar a cabo el tercer incremento, es decir, la adaptación de la herramienta a un sistema web. También se considera parte del mantenimiento adaptativo la incorporación de una pasarela de pago, planificado para el segundo incremento, que se sincronice con la interfaz TPV implementada en la herramienta.

6.2.4. MANTENIMIENTO PERFECTIVO

El mantenimiento perfectivo consiste en realizar las acciones de mejora en el sistema necesarias para cumplir con posibles nuevas necesidades impuestas por el cliente después de la entrega del producto.

En nuestro proyecto, este mantenimiento engloba las actividades que se realizarán durante el desarrollo del segundo incremento, en el cual, como hemos visto en el apartado 6.1, se basa en la incorporación de nuevas funcionalidades al sistema.

7. CONCLUSIONES

La realización de este proyecto me ha permitido poner en práctica los conocimientos adquiridos durante la carrera en asignaturas como *Estructuras de Datos*, *Sistemas Informáticos*, *Análisis y Desarrollo de Software* (y proyecto) o *Ingeniería del Software* (y proyecto). Dichos conocimientos se han empleado en un caso real y sin el amparo del marco de trabajo que los enunciados de las prácticas realizadas durante el grado nos ofrecían. Resulta mucho más complejo partir de una idea y ser capaz, mediante intensas jornadas de reflexión, de determinar qué hay que hacer y cómo deber hacerse, que preguntarse qué me piden que haga y cómo quieren que lo haga.

Otro factor a tener en cuenta, consiste en la dificultad que supone trabajar con un cliente de verdad. Resulta muy complejo, y en ocasiones puede ser frustrante, transmitir las ideas y conocimientos que tienes en la cabeza a una persona completamente ajena a la ingeniería informática, y no queda más remedio que armarse de paciencia para tratar de explicar, las veces que hagan falta, tus ideas u opiniones de una manera que el cliente sea capaz de comprender. En los momentos de bloqueo se ha recurrido a emplear explicaciones visuales mediante dibujos en papel o pizarra.

Además, he podido ampliar y consolidar los conocimientos hasta ahora adquiridos en el diseño de bases de datos, uso de patrones y metodologías de desarrollo, poniendo en práctica los conocimientos ya adquiridos en la carrera, y empleando nuevos conocimientos para mí como el uso de MyBatis como framework de persistencia o la puesta en práctica de una metodología de desarrollo cercana a ATDD (Aceptance Test-Driven Development).

También he podido verificar la importancia de llevar una documentación (aunque sea de manera informal) de absolutamente todo lo que vayas realizando a lo largo del proyecto, ya que esto nos permite relacionar, de una manera mucho más sencilla, las diferentes actividades que vamos realizando y nos ayuda a no caer en errores típicos como analizar las mismas ideas varias veces u olvidarnos de aspectos de diseño importantes a tener en cuenta en futuras decisiones. Esta metodología no estamos acostumbrados a emplearla en las prácticas realizadas durante la carrera debido a que la ventana temporal en la que se desarrollan las mismas es mucho menor, y no nos permite darnos cuenta de la importancia que tiene documentar el trabajo realizado, para dentro de uno o dos meses, poder recordar el trabajo que se realizó de una manera fácil y rápida.

Otra de las conclusiones que he podido extraer tras la realización de este trabajo, ha sido la importancia de la modularidad en las aplicaciones software. La división en módulos dota de una gran flexibilidad al sistema, permitiendo que el desarrollo se pueda realizar por personas independientes de manera muy sencilla. Además, facilita enormemente las tareas de mantenimiento de la aplicación, ya que el sistema, aunque probablemente sea más grande y albergue más líneas de código que empleando otro tipo de metodologías, se vuelve mucho más limpio y claro para la persona que tenga que trabajar con él, y permitiendo, en caso de que sea necesario, cambiar un módulo del sistema sin tener que realizar casi ningún cambio en el resto de módulos.

Como conclusión final, cabe destacar la tremenda satisfacción personal que produce el saber diseñar desde cero una herramienta, según el cliente muy util, sin necesitar nada más que tiempo, ganas e ilusión. Respecto a la valoración final del cliente, cito sus palabras textuales: *"es la adquisición más rentable que he realizado para mi negocio en mucho tiempo"*.

8. ANEXOS

ANEXO A. PLANTILLA DEL MODELO DE PROCESO CONVENCIONAL

Valores de los criterios de personal	**Convencional**			Puntuación
(c1) Experiencia del usuario	0	0	1	0
(c2) Expresividad del usuario	0	0	1	0
(c3) Experiencia del desarrollador en el ámbito de la aplicación	0	0	1	0
(c4) Experiencia del desarrollador en el entorno SW	1	1	1	1
Valores de los criterios del problema				
(c5) Madurez de la aplicación	0	0	1	0
(c6) Complejidad del problema	1	0	0	0
(c7) Requerimientos de funcionalidad parcial	1	0	0	0
(c8) Frecuencia de cambios	1	0	0	0
(c9) Magnitud de los cambios	1	0	0	1
Valores de los criterios del producto				
(c10) Tamaño del producto	1	0	0	0
(c11) Complejidad del producto	1	0	0	1
(c12) Requerimientos cualitativos	1	1	1	1
(c13) Requerimientos de interfaz de usuario	1	0	0	0
Valores de los criterios del recurso				
(c14) Financiación	1	0	0	-
(c15) Disponibilidad de fondos	0	0	1	-
(c16) Perfil del personal	1	0	0	0
(c17) Disponibilidad de personal	0	0	1	-
(c18) Accesibilidad a los usuarios	0	1	1	0
Valores de los criterios organizacionales				
(c19) Gestión de compatibilidad	1	1	1	1
(c20) Garantía de calidad y metodología administrativa	1	1	1	-
Total				**5**

Valores de los criterios de personal	**Incremental**			Puntuación
(c1) Experiencia del usuario	0	1	1	0
(c2) Expresividad del usuario	0	1	1	1
(c3) Experiencia del desarrollador en el ámbito de la aplicación	0	1	1	1
(c4) Experiencia del desarrollador en el entorno SW	0	1	1	1
Valores de los criterios del problema				
(c5) Madurez de la aplicación	0	1	1	1
(c6) Complejidad del problema	1	1	0	1
(c7) Requerimientos de funcionalidad parcial	0	1	0	0
(c8) Frecuencia de cambios	1	1	0	1
(c9) Magnitud de los cambios	1	1	0	1
Valores de los criterios del producto				
(c10) Tamaño del producto	1	1	1	1
(c11) Complejidad del producto	1	1	1	1
(c12) Requerimientos cualitativos	1	1	0	1
(c13) Requerimientos de interfaz de usuario	1	1	0	1
Valores de los criterios del recurso				
(c14) Financiación	0	1	0	-
(c15) Disponibilidad de fondos	0	1	1	-
(c16) Perfil del personal	0	1	0	1
(c17) Disponibilidad de personal	0	1	1	-
(c18) Accesibilidad a los usuarios	0	1	1	0
Valores de los criterios organizacionales				
(c19) Gestión de compatibilidad	1	1	0	1
(c20) Garantía de calidad y metodología administrativa	0	1	1	-
Total				**13**

Valores de los criterios de personal	**Evolutivo**			Puntuación
(c1) Experiencia del usuario	1	1	1	1
(c2) Expresividad del usuario	1	1	1	1
(c3) Experiencia del desarrollador en el ámbito de la aplicación	1	1	1	1
(c4) Experiencia del desarrollador en el entorno SW	0	1	1	1
Valores de los criterios del problema				
(c5) Madurez de la aplicación	1	1	1	1
(c6) Complejidad del problema	1	1	1	1
(c7) Requerimientos de funcionalidad parcial	0	1	1	0
(c8) Frecuencia de cambios	1	1	1	1
(c9) Magnitud de los cambios	1	1	1	1
Valores de los criterios del producto				
(c10) Tamaño del producto	1	1	1	1
(c11) Complejidad del producto	1	1	0	0
(c12) Requerimientos cualitativos	1	0	0	0
(c13) Requerimientos de interfaz de usuario	1	1	1	1
Valores de los criterios del recurso				
(c14) Financiación	0	1	0	-
(c15) Disponibilidad de fondos	1	1	1	-
(c16) Perfil del personal	0	1	0	1
(c17) Disponibilidad de personal	1	1	1	-
(c18) Accesibilidad a los usuarios	0	0	1	0
Valores de los criterios organizacionales				
(c19) Gestión de compatibilidad	1	1	0	1
(c20) Garantía de calidad y metodología administrativa	0	1	1	-
Total				**12**

Valores de los criterios de personal	**Cascada**			Puntuación
(c1) Experiencia del usuario	0	0	1	0
(c2) Expresividad del usuario	0	0	1	0
(c3) Experiencia del desarrollador en el ámbito de la aplicación	0	1	1	1
(c4) Experiencia del desarrollador en el entorno SW	1	1	1	1
Valores de los criterios del problema				
(c5) Madurez de la aplicación	0	0	1	0
(c6) Complejidad del problema	1	0	0	0
(c7) Requerimientos de funcionalidad parcial	0	0	0	0
(c8) Frecuencia de cambios	1	0	0	0
(c9) Magnitud de los cambios	0	0	0	0
Valores de los criterios del producto				
(c10) Tamaño del producto	0	0	0	0
(c11) Complejidad del producto	1	0	0	1
(c12) Requerimientos cualitativos	0	0	0	0
(c13) Requerimientos de interfaz de usuario	1	0	0	1
Valores de los criterios del recurso				
(c14) Financiación	1	0	0	-
(c15) Disponibilidad de fondos	0	0	0	-
(c16) Perfil del personal	1	0	0	0
(c17) Disponibilidad de personal	0	0	0	-
(c18) Accesibilidad a los usuarios	0	1	1	1
Valores de los criterios organizacionales				
(c19) Gestión de compatibilidad	1	1	1	1
(c20) Garantía de calidad y metodología administrativa	1	1	1	-
Total				**6**

Valores de los criterios de personal	**Híbrido**			Puntuación
(c1) Experiencia del usuario	1	1	1	1
(c2) Expresividad del usuario	1	1	1	1
(c3) Experiencia del desarrollador en el ámbito de la aplicación	1	1	1	1
(c4) Experiencia del desarrollador en el entorno SW	0	1	1	1
Valores de los criterios del problema				
(c5) Madurez de la aplicación	1	1	1	1
(c6) Complejidad del problema	1	1	1	1
(c7) Requerimientos de funcionalidad parcial	0	0	0	0
(c8) Frecuencia de cambios	1	1	0	0
(c9) Magnitud de los cambios	0	0	0	0
Valores de los criterios del producto				
(c10) Tamaño del producto	0	0	0	0
(c11) Complejidad del producto	1	1	1	1
(c12) Requerimientos cualitativos	0	0	0	0
(c13) Requerimientos de interfaz de usuario	1	1	1	1
Valores de los criterios del recurso				
(c14) Financiación	0	1	0	-
(c15) Disponibilidad de fondos	0	0	0	-
(c16) Perfil del personal	0	1	0	1
(c17) Disponibilidad de personal	0	0	0	-
(c18) Accesibilidad a los usuarios	0	0	1	0
Valores de los criterios organizacionales				
(c19) Gestión de compatibilidad	1	1	0	1
(c20) Garantía de calidad y metodología administrativa	0	1	1	-
Total				**10**

Valores de los criterios de personal	**Operacional**			Puntuación
(c1) Experiencia del usuario	0	1	1	0
(c2) Expresividad del usuario	0	1	1	1
(c3) Experiencia del desarrollador en el ámbito de la aplicación	0	1	1	1
(c4) Experiencia del desarrollador en el entorno SW	0	1	1	1
Valores de los criterios del problema				
(c5) Madurez de la aplicación	0	1	1	1
(c6) Complejidad del problema	1	1	0	1
(c7) Requerimientos de funcionalidad parcial	0	0	0	0
(c8) Frecuencia de cambios	1	1	0	0
(c9) Magnitud de los cambios	0	0	0	0
Valores de los criterios del producto				
(c10) Tamaño del producto	0	0	0	0
(c11) Complejidad del producto	1	0	0	1
(c12) Requerimientos cualitativos	0	0	0	0
(c13) Requerimientos de interfaz de usuario	1	1	0	1
Valores de los criterios del recurso				
(c14) Financiación	0	1	0	-
(c15) Disponibilidad de fondos	0	0	0	-
(c16) Perfil del personal	0	1	0	1
(c17) Disponibilidad de personal	0	0	0	-
(c18) Accesibilidad a los usuarios	0	0	1	0
Valores de los criterios organizacionales				
(c19) Gestión de compatibilidad	1	0	0	0
(c20) Garantía de calidad y metodología administrativa	0	1	1	-
Total				**8**

Aspectos generales de las ventanas.

Como se puede observar en la **Figura 48**, todas las ventanas que componen la aplicación cuentan, en su parte superior, de un título que identifica la sección a la que pertenece la ventana, así como su nombre. Entre el nombre de sección y el nombre de la ventana, se encuentra un botón circular de color verde que permite volver al menú de la sección a la que pertenezca la ventana.

Pantalla de inicialización y bienvenida.

Al arrancarse, la herramienta muestra una pantalla inicial con un diálogo de espera mientras que en segundo plano se comprueba si, desde la última vez que se inició, se debe registrar algún pago programado de un gasto fijo, si se ha actualizado el registro histórico del stock o si se han añadido al histórico de contabilidad los registros correspondientes.

FIGURA 36. *PANTALLA DE INICIALIZACIÓN.*

Una vez finalizadas las tareas en segundo plano, se solicita identificación al usuario.

Autentificación.

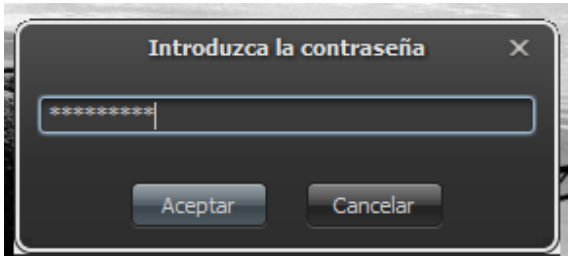

FIGURA 37. *AUTENTIFICACIÓN.*

Si se realiza correctamente la autentificación, se da paso al menú inicial.

Menú inicial.

Consta de 4 botones ilustrados, uno para cada sección de la aplicación.

FIGURA 38. *MENÚ INICIAL.*

Menú movimientos rápidos.

Da acceso a las nuevas ventas, devoluciones y pedidos. Para editar datos sobre registros ya existentes se emplea un cuarto botón que nos lleva a un formulario.

FIGURA 39. *MENÚ MOVIMIENTOS RÁPIDOS.*

Menú de informes.

Esta sección no se ha implementado para este primer incremento, pero será la encargada de dar acceso a los informes que genere la aplicación tales como balances, cuentas de pérdidas y ganancias en diferentes rangos temporales (informes diarios, mensuales, trimestrales y anuales) o informes de rendimientos de productos y clientes.

Menú gestión datos.

Acceso a los diferentes formularios que permiten, para cada entidad gestionada en la aplicación, crear nuevos registros, realizar consultas y actualizar o borrar los registros existentes. Dichos formularios contienen motores de búsqueda que permiten, mediante listas desplegables (con funcionalidad extra de autocompletado) y selectores de fechas, realizar un filtrado de los datos a visualizar de una manera muy sencilla.

FIGURA 40. *MENÚ DE GESTIÓN DE DATOS.*

Menú de configuración.

Permite personalizar diferentes aspectos de configuración de la aplicación, tales como permitir la actualización automática de los pagos ocasionados por los gastos fijos, cambiar el tema de la interfaz, cambiar la contraseña de acceso, configurar el directorio de salida de informes o configurar el directorio donde se encuentran las imágenes de los productos.

FIGURA 41. *MENÚ DE CONFIGURACIÓN DE LA APLICACIÓN.*

Formularios.

Las ventanas empleadas para la gestión de datos y la configuración de la aplicación tienen un diseño a modo de formulario.

Los formularios de gestión de datos se caracterizan en su mayoría por tener en su parte superior una tabla de recogida de los datos a insertar en el nuevo registro, debajo un menú con los diferentes filtros para el motor de búsqueda y a continuación la tabla de resultados a mostrar. En algunos casos, a este diseño se la añade un menú inferior de navegación rápida que permite navegar de manera inmediata a otro formulario que tenga relación con el activo.

FIGURA 42. *SECCIÓN NUEVO REGISTRO.*

Para los datos cuya validación reside en comprobar el tipo de dato introducido se emplea una tabla de una única fila que permite la edición del registro, mostrando en rojo el campo incorrecto (**Figura 43**).

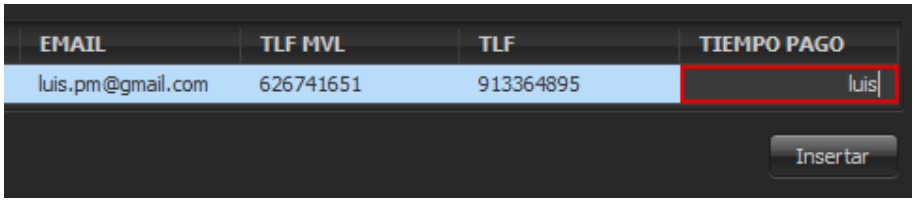

FIGURA 43. *ERROR VALIDANDO EL CAMPO TIEMPO PAGO.*

Si el error cometido a la hora de insertar no se produce por un error en el tipo de datos introducido, se muestra un mensaje de error como el de la **Figura 44**, que informa de un error al querer insertar un cliente con un código que ya existe en base de datos.

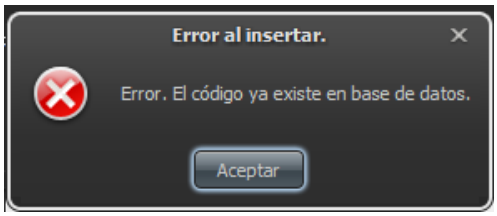

FIGURA 44. *ERROR POR CÓDIGO DUPLICADO.*

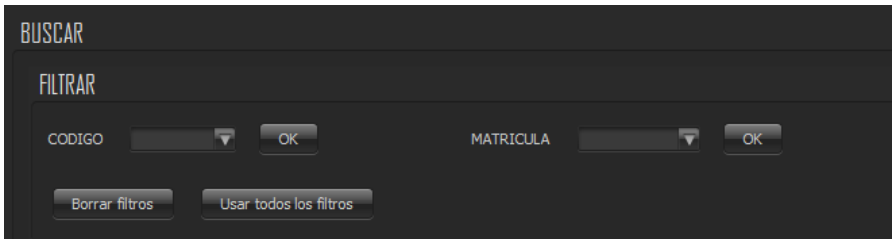

FIGURA 45. *FILTROS DEL BUSCADOR.*

La sección de búsqueda (**Figura 45**) nos ofrece 3 posibilidades:

- **OK:** nos permite buscar aplicando únicamente como criterio de búsqueda el valor seleccionado en ese filtro.
- **Usar todos los filtros:** realiza la búsqueda aplicando los valores seleccionados en todos los filtros ofrecidos (siempre y cuando tengan un valor seleccionado).
- **Borrar filtros:** hace un Reset de los valores seleccionados en los filtros y posteriormente ejecuta la búsqueda por defecto para el formulario en cuestión.

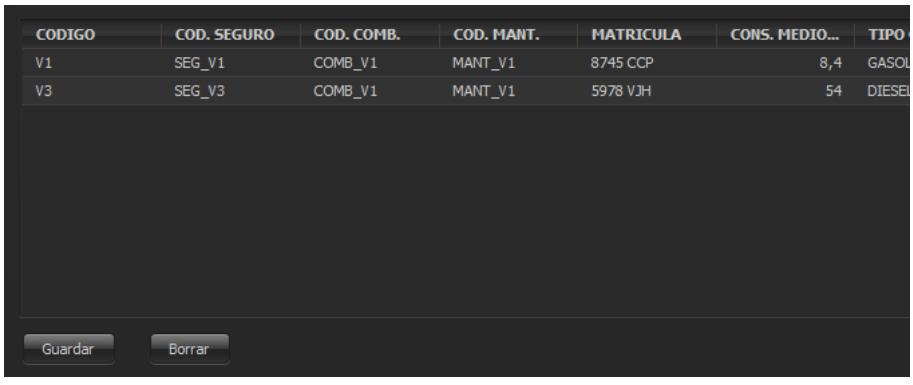

FIGURA 46. *TABLA DE RESULTADOS.*

Los resultados de las búsquedas realizadas se muestran en una tabla como la de la **Figura 46**, mediante la cual, además de consultar datos, podemos editarlos o borrarlos. En el caso de editar, sólo se podrán guardar los registros de uno en uno (se guarda el último editado) para evitar tener que guardar toda la tabla si sólo se ha modificado un registro. En incrementos futuros del proyecto se permitirá guardar sólo los registros editados, los cuales se sombrearán en otro color para evitar que el usuario se olvide.

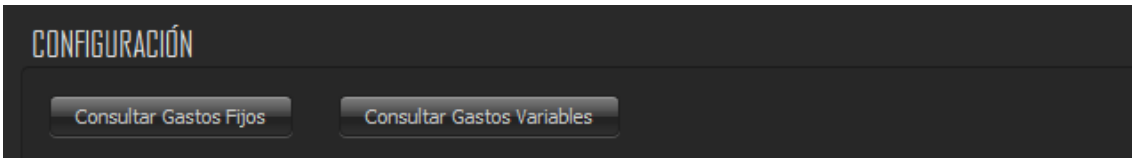

FIGURA 47. *MENÚ NAVEGACIÓN RÁPIDA.*

El menú de navegación rápida (**Figura 47**) contiene botones de acceso inmediato a formularios que tienen una relación directa con el cuestionario actualmente activo.

En la **Figura 48** podemos contemplar el aspecto global de uno de los formularios de gestión de datos diseñado para la herramienta.

FIGURA 48. *FORMULARIO DE GESTIÓN DE VEHÍCULOS.*

Movimiento rápido: TPV.

El movimiento rápido de registro de una nueva venta, así como el de nuevo pedido, consta de una interfaz similar a la empleada típicamente en los TPV. El término TPV lo aplicamos solamente a la interfaz desarrollada, ya que dicha ventana no cuenta con un sistema de cobro con el que esté sincronizado.

FIGURA 49. *INTERFAZ TPV NUEVA VENTA.*

En primer lugar, recoge los datos del cliente (**Figura 50**), ya sea registrado o anónimo.

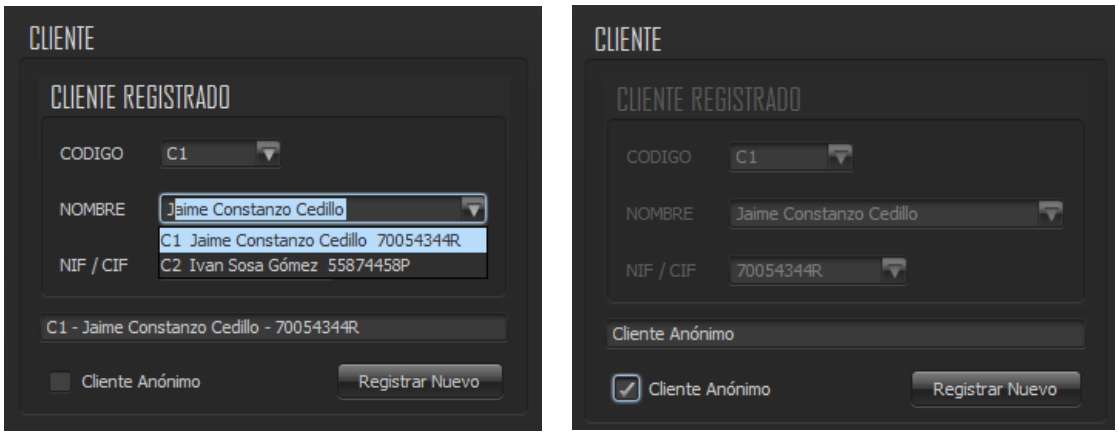

FIGURA 50. *SELECCIÓN DE UN CLIENTE REGISTRADO / ANÓNIMO.*

A continuación, podemos seleccionar los productos que se deseen adquirir, pudiendo escoger entre el área de construcción o el de jardinería. Cuando se selecciona un producto, se pregunta el tipo de envasado (**Figura 51**) a seleccionar entre los disponibles y se inserta en el "carrito de la compra" o "panel del pedido" (**Figura 52**), desde el cual se podrá modificar el número de unidades vendidas del producto, así como el precio de venta de cada unidad y la cuantía del descuento que se pudiese realizar.

FIGURA 51. *SELECCIÓN ENVASADO PRODUCTO.*

FIGURA 52. *PANEL DEL PEDIDO ACTUAL.*

En un futuro, los botones que representan los productos estarán ilustrados con una imagen del propio producto.

Si al seleccionar un producto hemos seleccionado previamente un cliente, se aplicará el precio específico de venta de ese producto para ese cliente. En caso de no existir una relación guardada, se empleará el precio por defecto del producto y se

guardará como un nuevo registro de la relación el precio que se aplique en la venta realizada.

Por último, decir que los productos, como vemos en la **Figura 53**, se muestran organizados por categorías y, de existir, por subcategorías a diferentes niveles.

FIGURA 53. *CATEGORÍAS Y SUBCATEGORÍAS JARDINERÍA.*

ANEXO H. CUESTIONARIO DE USABILIDAD.

PREGUNTAS	Muy Bajo	Bajo	Medio	Alto	Muy Alto
1. Facilidad para saber en todo momento en qué parte de la aplicación estamos.	1	2	3	4	5
2. Facilidad de comprensión del lenguaje empleado en la aplicación.	1	2	3	4	5
3. La información se muestra en un orden lógico que facilita su entendimiento.	1	2	3	4	5
4. Los menús de la aplicación son sencillos de utilizar y fáciles de comprender.	1	2	3	4	5
5. Es fácil y rápido registrar ventas y pedidos o realizar cualquier otra acción que resulte crítica.	1	2	3	4	5
6. Facilidad para corregir los errores cometidos en la introducción de datos al sistema.	1	2	3	4	5
7. Los botones y demás elementos visuales siguen un mismo estilo que facilita su uso.	1	2	3	4	5
8. La aplicación emplea herramientas que reducen la posibilidad de cometer errores al introducir datos.	1	2	3	4	5
9. El usuario no necesita recordar información que recibió previamente.	1	2	3	4	5
10. El sistema permite evitar tener que repetir la misma información para realizar las mismas acciones.	1	2	3	4	5
11. Los mensajes de error son muy claros y concisos, explicando exactamente el problema producido.	1	2	3	4	5

TABLA 5. *CUESTIONARIO DE USABILIDAD PARA EL USUARIO.*

ANEXO I. PRESUPUESTO

Aunque en nuestro caso el coste del proyecto para el cliente ha sido cero, en esta sección desglosamos un presupuesto con los posibles costes de desarrollo del sistema.

1. Honorarios Proyecto

- Incremento 1: 500 horas
  - Analista: 100 horas a 30 € / hora — 3000 €
  - Diseñador: 150 horas a 30 € / hora — 4500 €
  - Desarrollador: 200 horas a 20 € / hora — 4000 €
  - Tester: 50 horas a 20 € / hora — 1000 €

- Incremento 2: 400 horas
  - Analista: 75 horas a 30 € / hora — 2250 €
  - Diseñador: 100 horas a 30 € / hora — 3000 €
  - Desarrollador: 150 horas a 20 € / hora — 3000 €
  - Tester: 75 horas a 20 € / hora — 1500 €

- Incremento 3: 450 horas
  - Analista: 100 horas a 30 € / hora — 3000 €
  - Diseñador: 100 horas a 30 € / hora — 3000 €
  - Desarrollador: 200 horas a 20 € / hora — 4000 €
  - Tester: 50 horas a 20 € / hora — 1000 €

2. Subtotal del presupuesto

- Subtotal Presupuesto
  - Incremento 1: 12500 €
  - Incremento 2: 9750 €
  - Incremento 3: 11000 €
  - Total: 33250 €

3. I.V.A. aplicable

- 21% Subtotal Presupuesto ..... 6982.5 €

4. Total presupuesto

- Total Presupuesto ..... 40232,5 €

Madrid, julio de 2014 El Ingeniero Jefe de Proyecto

REFERENCIAS

1. Plan General Contable. http://www.plangeneralcontable.com/?tit=cuadro-de-cuentas &name=Abanfin&fid=pgc0005 [Consultado el: 08 de mayo de 2014].

2. **Alexander, L. and Davis, A.** *Criteria for Selecting Software Process Models.* Vols. Proceedings of the Fifteenth Annual International Computer Software and Applications Conference (COMPSAC), IEEE 1991.

3. **Sommerville, Ian.** *Ingeniería de Software.* 7ª. s.l. : Pearson, 2005.

4. **Pressman, R.** *Ingeniería del Software. Un Enfoque Práctico.* 6ª. s.l. : McGraw Hill, 2005.

5. **Sri Manoj Kumar Tyagi, M.Pushpa Rekha, K. Vamsikrishnareddy, A. Sudhakar.** *Advances in Mechanical Engineering and its Applications.* United States : World Science Publisher, 2012.

6. *ISO/IEC 14764:2006. IEEE Std 14764-2006.*

7. **Bushmann F., Meunier R., Rohnert H., Sommerlad P., Stal M.** *Pattern-Oriented Software Architecture: A System of Patterns.* New York : Wiley, 1996.

8. **Gamma E., Helm R., Johnson R., Vlissides J.** *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, Massachusetts : Addison-Wesley, 1995.

9. **Alur D., Crupi J., Malks D.** *Core J2EE Patterns: Best Practices and Design Strategies.* Englewood Cliffs, New Jersey : Prentice-Hall, 2001.

10. [**Canarias, Iker]** slideshare.net. http://www.slideshare.net/ikercanarias/persistencia-de-datos-en-java. [Consultado el: 04 de enero de 2014].

11. Nosolounix.com. http://www.nosolounix.com/2010/03/programacion-java-la-capa-de.html. [Consultado el: 16 de enero de 2014].

12. MyBatis.org. http://mybatis.github.io/mybatis-3/es/. [Consultado el: 06 de enero de 2014].

13. **Reddy, K. Siva Prasad.** *Java Persistence with MyBatis 3.* s.l. : Packt Publishing, 2013.

14. [**Nielsen, Norman]** Nielsen Norman Group. http://www.nngroup.com/articles/ten-usability-heuristics. [Consultado el: 12 de marzo de 2014].

15. Construmática. http://www.construmatica.com/construpedia/Mantenimiento_ Correctivo [Consultado el: 8 de abril de 2013].

16. Wikipedia. http://es.wikipedia.org/wiki/Mantenimiento_correctivo. [Consultado el: 8 de abril de 2013].

17. jummp.wordpress.com. http://jummp.wordpress.com/2010/09/25/mantenimiento-adaptativo/ [Consultado el: 10 de abril de 2013].